The Dissertation Committee for Nathan Vanderkooy Roberts
certifies that this is the approved version of the following dissertation:

A Discontinuous Petrov-Galerkin Methodology for Incompressible Flow Problems

Committee:

Leszek Demkowicz, Supervisor

Robert Moser, Co-Supervisor

Todd Arbogast

George Biros

Thomas Hughes

A Discontinuous Petrov-Galerkin Methodology for Incompressible Flow Problems

by

Nathan Vanderkooy Roberts, B.A.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2013

Dedicated to my parents, Bob and Elizabeth Roberts.

Acknowledgments

I have been supported by the Department of Energy [National Nuclear Security Administration] under Award Number [DE-FC52-08NA28615]. I would like to thank my advisors, Leszek Demkowicz and Robert Moser, and the rest of my thesis committee—Todd Arbogast, George Biros, and Thomas Hughes—for their support and guidance of my work. I also thank Pavel Bochev and Denis Ridzal for hosting and collaborating with me in the summers of 2010 and 2011, when I was supported by internships at Sandia National Laboratories (Albuquerque). I thank Tan Bui-Thanh and Jesse Chan for their continuing collaboration in various aspects of my research.

Provenance. Some of the material in this document was originally developed in collaboration with others, some of which has been published in other contexts. Specifically, the discussion of distributing the stiffness matrix computation in Appendix A was written with Jesse Chan (but not published); portions of Chapters 1, 4, and 6 were written with Tan Bui Thanh and Leszek Demkowicz [72]; some of the material regarding development and verification of Camellia in Appendix B arose from a collaboration with Denis Ridzal and Pavel Bochev [73].

A Discontinuous Petrov-Galerkin Methodology for Incompressible Flow Problems

Publication No. _____

Nathan Vanderkooy Roberts, Ph.D.
The University of Texas at Austin, 2013

Supervisors: Leszek Demkowicz
Robert Moser

Incompressible flows—flows in which variations in the density of a fluid are negligible—arise in a wide variety of applications, from hydraulics to aerodynamics. The incompressible Navier-Stokes equations which govern such flows are also of fundamental physical and mathematical interest. They are believed to hold the key to understanding turbulent phenomena; precise conditions for the existence and uniqueness of solutions remain unknown—and establishing such conditions is the subject of one of the Clay Mathematics Institute's Millennium Prize Problems.

Typical solutions of incompressible flow problems involve both fine- and large-scale phenomena, so that a uniform finite element mesh of sufficient granularity will at best be wasteful of computational resources, and at worst be infeasible because of resource limitations. Thus adaptive mesh refinements are required. In industry, the adaptivity schemes used are ad hoc, requiring

a domain expert to predict features of the solution. A badly chosen mesh may cause the code to take considerably longer to converge, or fail to converge altogether. Typically, the Navier-Stokes solve will be just one component in an optimization loop, which means that any failure requiring human intervention is costly.

Therefore, I pursue technological foundations for a solver of the incompressible Navier-Stokes equations that provides robust adaptivity starting with a coarse mesh. By robust, I mean both that the solver always converges to a solution in predictable time, and that the adaptive scheme is independent of the problem—no special expertise is required for adaptivity.

The cornerstone of my approach is the discontinuous Petrov-Galerkin (DPG) finite element methodology developed by Leszek Demkowicz and Jay Gopalakrishnan. For a large class of problems, DPG can be shown to converge at optimal rates. DPG also provides an accurate mechanism for measuring the error, and this can be used to drive adaptive mesh refinements.

Several approximations to Navier-Stokes are of interest, and I study each of these in turn, culminating in the study of the steady 2D incompressible Navier-Stokes equations. The Stokes equations can be obtained by neglecting the convective term; these are accurate for "creeping" viscous flows. The Oseen equations replace the convective term, which is nonlinear, with a linear approximation. The steady-state incompressible Navier-Stokes equations approximate the transient equations by neglecting time variations.

Crucial to this work is *Camellia*, a toolbox I developed for solving DPG problems which uses the Trilinos numerical libraries. Camellia supports 2D meshes of triangles and quads of variable polynomial order, allows simple specification of variational forms, supports h- and p-refinements, and distributes the computation of the stiffness matrix, among other features.

The central contribution of this dissertation is design and development of mathematical techniques and software, based on the DPG method, for solving the 2D incompressible Navier-Stokes equations in the laminar regime (Reynolds numbers up to about 1000). Along the way, I investigate approximations to these equations—the Stokes equations and the Oseen equations—followed by the steady-state Navier-Stokes equations.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Motivation. Incompressible flows—flows in which variations in the density of a fluid are negligible—arise in a wide variety of applications, from hydraulics to aerodynamics. The incompressible Navier-Stokes equations which govern such flows are also of fundamental physical and mathematical interest. They are believed to hold the key to understanding turbulent phenomena; precise conditions for the existence and uniqueness of solutions remain unknown—and establishing such conditions is the subject of one of the Clay Mathematics Institute's Millennium Prize Problems.

Typical solutions of incompressible flow problems involve both fine- and large-scale phenomena, so that a uniform finite element mesh of sufficient granularity will at best be wasteful of computational resources, and at worst be infeasible because of resource limitations. Thus adaptive mesh refinements are required. In industry, the adaptivity schemes used are ad hoc, requiring a domain expert to predict features of the solution. A badly chosen mesh may cause the code to take considerably longer to converge, or fail to converge altogether. Typically, the Navier-Stokes solve will be just one component in an

optimization loop, which means that any failure requiring human intervention is costly.

Goal. Our aim is to develop a solver for the incompressible Navier-Stokes equations that provides robust adaptivity starting with a coarse mesh. By robust, we mean both that the solver always converges to a solution in predictable time, and that the adaptive scheme is independent of the problem—no special expertise is required for adaptivity.

The cornerstone of our approach will be the discontinuous Petrov-Galerkin with optimal test functions (DPG) finite element methodology recently developed by Leszek Demkowicz and Jay Gopalakrishnan [35, 37]. Whereas Bubnov-Galerkin methods use the same function space for both test and trial functions, Petrov-Galerkin methods allow the spaces for test and trial functions to differ. In DPG, the test functions are computed on the fly and are chosen to minimize the residual. For a very broad class of well-posed problems, DPG offers provably optimal convergence rates with a modest stability constant—the "inf-sup" constants governing the convergence are mesh-independent, and of the same order as those governing the continuous problem [72]. In some of our experiments, DPG not only achieves the optimal rates, but gets very close to the best solution available in the discrete space. DPG also provides an accurate mechanism for measuring the error, and this can be used to drive adaptive mesh refinements.

Several approximations to Navier-Stokes are of physical interest, and

our approach will involve studying each of these in turn, culminating in the study of the 2D incompressible Navier-Stokes equations. The Stokes equations can be obtained by neglecting the convective term; these are accurate for "creeping" viscous flows. The Oseen equations replace the convective term, which is nonlinear, with a linear approximation. The steady-state incompressible Navier-Stokes equations can be obtained by setting the time derivatives in the transient equations to zero.

We have studied DPG applied to the Stokes problem in some detail, with theoretical results predicting optimal rates of convergence, and numerical results that appear to show even more: it appears that we asymptotically approach the best approximation error available in the discrete space [72]. Because of this success and the close relationship between the Stokes equations and the incompressible Navier-Stokes equations, we are optimistic that we can achieve good results with the latter as well.

Central to our study of these problems has been the use and further development of *Camellia* [73], a toolbox we developed for solving DPG problems which uses Sandia's Trilinos library of packages [56]. I began work on Camellia in collaboration with Denis Ridzal and Pavel Bochev during an internship at Sandia. At present, Camellia supports 2D meshes of triangles and quads of variable polynomial order, provides mechanisms for easy specification of DPG variational forms, supports h- and p- refinements, and supports distributed computation of the stiffness matrix, among other features. In the future, we hope to add support for meshes of arbitrary spatial dimension,

support for space-time elements, and support for distributed mesh and solution representation.

Structure of the Dissertation. The remainder of this introduction is structured as follows. In Section 1.2, we review some relevant approaches in the literature on incompressible flow; in Section 1.3, we discuss three popular FEM codes (`deal.II`, libMesh, and FEniCS), for later comparison and contrast with Camellia. In Section 1.4, we describe some features of DPG and how we propose to exploit these in the context of incompressible flow problems. In Section 1.5, we describe the proposed approach to designing FEM software that implements the DPG method (Camellia). We conclude this chapter in Section 1.6 with a discussion of the contributions of the dissertation.

The main body of the dissertation is organized into three parts. In the first, we describe the execution of a DPG solve for a given PDE problem and its corresponding implementation in Camellia. In Chapter 2, we discuss the mathematical and computational steps required to use DPG to solve a PDE, taking the velocity-pressure-gradient (VGP) Stokes formulation as a reference example; there, we also derive two other Stokes formulations, the velocity-stress-pressure (VSP) and velocity-vorticity-pressure (VVP) formulations. We continue in Chapter 3 with the steps required to implement a DPG solver using Camellia.

Next, we analyze of DPG as a methodology and verify the implementation of Camellia—providing warrant for belief that the steps previously

described will yield desirable results. In Chapter 4, we analyze DPG, again using the VGP Stokes formulation as our touchstone. In Chapter 5, we describe some of the steps we have taken to verify Camellia's code.

We then turn to our work applying DPG to the various incompressible flow problems. In Chapter 6, we perform a host of numerical experiments using the VGP Stokes formulation. In Chapter 7, we develop a DPG formulation for the Oseen equations inspired by the VGP Stokes formulation, and perform some numerical experiments to verify it. In Chapter 8, we similarly develop a formulation for the Navier-Stokes equations, and perform a wide variety of numerical experiments to examine the potential of DPG in this context.

In Chapter 9, we examine the conditioning of two matrices involved in the solution of DPG problems—the local Gram matrix and the global stiffness matrix—and the effect of various choices (test space norm, test space basis, and the use of static condensation in the global solve) on that conditioning.

We conclude in Chapter 10 with a recapitulation of the contributions of this dissertation and a discussion of possible future work extending from it. Appendix A documents our approach to distributing the computation of the stiffness matrix in Camellia, and Appendix B provides some additional documentation of tests we have run to verify Camellia.

1.2 Selected Literature Review: Incompressible Flow Problems

In this section, we review selected literature regarding numerical solutions of the Stokes, Oseen, and incompressible Navier-Stokes equations. Some of the principal challenges of solving incompressible Navier-Stokes—in particular, its saddle-point character—can already be seen in the Stokes problem, and finite element discretizations and analysis for the Stokes problem often apply to Navier-Stokes as well. Note that by necessity this review is far from comprehensive—instead, we highlight a few key historical developments and recent methods of interest, particularly those that relate closely in one way or another to our present approach.

First, though, we briefly state the incompressible transient Navier-Stokes equations, and each of the approximations to them that we will investigate. The Navier-Stokes equations are:

$$\begin{aligned} -\nabla p + \mu \Delta \boldsymbol{u} &= \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \boldsymbol{g}, \\ \nabla \cdot \boldsymbol{u} &= 0, \end{aligned}$$

where p is the pressure, $\boldsymbol{u}$ is the velocity, $\boldsymbol{g}$ a vector forcing function (gravity, e.g.), and $\mu = \frac{1}{\mathrm{Re}}$ is the viscosity, assumed constant. The first equation corresponds to the conservation of momentum; the second to conservation of mass. We have taken density ρ to be a constant, non-dimensionalized to equal 1. From here forward we will neglect gravitational effects, taking $\boldsymbol{g} = \boldsymbol{0}$. These are the transient equations; with the steady-state assumption $\frac{\partial \boldsymbol{u}}{\partial t} = 0$

we get the steady equations:

$$-\nabla p + \mu \Delta \boldsymbol{u} = \boldsymbol{u} \cdot \nabla \boldsymbol{u},$$
$$\nabla \cdot \boldsymbol{u} = 0.$$

These are nonlinear thanks to the convective term $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$; if we approximate this by a linear term $\boldsymbol{U} \cdot \nabla \boldsymbol{u}$, where $\boldsymbol{U}$ is the free-stream velocity, we have

$$-\nabla p + \mu \Delta \boldsymbol{u} = \boldsymbol{U} \cdot \nabla \boldsymbol{u},$$
$$\nabla \cdot \boldsymbol{u} = 0,$$

which are the Oseen equations. If we neglect the convective term altogether we have

$$-\nabla p + \mu \Delta \boldsymbol{u} = \boldsymbol{0},$$
$$\nabla \cdot \boldsymbol{u} = 0,$$

the Stokes equations with zero forcing function.

Stokes and Oseen. The Stokes equations model incompressible viscous ("creeping") flow; they can be derived by neglecting the convective term in the incompressible Navier-Stokes equations. Naive discretizations for the Stokes problem can lead to non-convergence or locking [8]. Of crucial importance for Bubnov-Galerkin formulations of the Stokes equations—and more generally, of saddle point problems—is the satisfaction of the two so-called Brezzi inf-sup

conditions [15]. For the Stokes equations, the first of these, the "inf-sup in the kernel" condition, is satisfied automatically. If the discrete spaces for velocity $\boldsymbol{u}$ and pressure p are $\boldsymbol{V}_h \subset \boldsymbol{H}^1$ and $Q_h \subset L^2$, respectively, the second Brezzi condition for Stokes is then

$$\inf_{q \in Q_h} \sup_{v \in V_h} \frac{(q, \nabla \cdot \boldsymbol{v})}{\|q\|_{L_0^2} \|\boldsymbol{v}\|_{\boldsymbol{H}^1}} \geq \gamma_h \geq \gamma_0 > 0.$$

In the context of Stokes, this condition is often called the Ladyzhenskaya-Babuška-Brezzi (LBB) condition, because Ladyzhenskaya is said[1] to have proved the continuous analog of the condition for the Stokes equations [62]; much of the challenge in solving Stokes lies in the selection of discrete spaces that satisfy this condition.

In [8], Boffi, Brezzi, and Fortin survey some choices for finite element discretizations to satisfy the LBB condition for the Stokes problem, among which are the MINI element, Crouzeix-Raviart element, and the class of $Q_k - P_{k-1}$ elements. Generalized Hood-Taylor elements can be shown to satisfy the condition under certain regularity constraints on the mesh. (Each of these elements generalizes to three-dimensional spaces as well.) It is worth noting that each of these elements uses a polynomial approximation for pressure of

[1] This common claim has recently been called into question, in a talk by Martin Costabel at MAFELAP 2013. It appears that the attribution originated when Jacques-Louis Lions made a comment to this effect to J. Tinsley Oden, after the latter gave a talk referring to this condition simply as the Babuška-Brezzi condition. Oden therefore started referring to it as the LBB condition, but at least in available editions of the Ladyzhenskaya text, the rumored proof does not appear. A proof for smooth domains appears in Girault and Raviart [50, pp. 32-37]; see recent work by Costabel and Dauge for analysis on more general domains [32].

one order lower than that used for velocity, so that the theoretical optimal convergence rate is lower for the pressure than it is for the velocity.

Cockburn et al. have applied the local discontinuous Galerkin (LDG) method [22] to the Stokes problem [27]; LDG derives its name from the local elimination of some variables (in the case of Stokes, the stresses)—by comparison with standard DG methods, the global solve in LDG involves only about half as many unknowns, a significant savings. By means of carefully chosen numerical fluxes, the LDG method can enforce conservation laws weakly element by element, in a locally conservative way. The method also allows one to choose spaces for the pressure and velocity independently, so that equal-order approximations can be used, but Cockburn et al. show that the convergence rate for pressure and stress will be of order one less than that for velocity. They numerically compare the efficiency of using lower-order approximations for pressure and stress with that of equal-order approximations, and conclude that in most cases the equal-order approximations are more efficient: although both choices yield the same *rate* of convergence, the lower-order approximation requires more degrees of freedom to achieve the same accuracy.

Cockburn et al. have also applied LDG to the Oseen equations by combining their LDG discretization of Stokes with a classical DG discretization of the convective term, with similarly good results—optimal-order convergence when the pressure space is discretized with polynomials of degree one less than that of the velocity, and the possibility of using equal-order approximations for all variables with improved efficiency [28, 30]. Their numerical experiments

demonstrate success with Reynolds numbers from 1 to 1000.

Building on work begun in Evans's PhD. thesis [44], Evans and Hughes have applied their divergence-free B-splines to the Stokes equations, with excellent results: using equal-order velocity and pressure spaces, local conservation is automatic by virtue of the divergence-free basis used for the velocity, and in their experiments, pressure and velocity both converge at optimal[2] rates [45].

Navier-Stokes. In incompressible flows, velocity and pressure are coupled by an incompressibility constraint, which leads to a saddle-point system that can be very sensitive to the discretization. Guermond et al. review *projection* methods for incompressible flows [53], which can also be viewed as fractional step methods, in that each time step is broken into partial steps. These are attractive because they decouple the velocity and the pressure: at each time step, two elliptic equations need to be solved in sequence. The methods fall into three broad classes: pressure-correction methods (such as that of Chorin [26] and Temam [76, 77]), velocity-correction methods (such as that of Orszag et al. [70] or that of Karniadakis et al. [58]), and consistent splitting methods such as that of Guermond and Shen [55]. The pressure- and velocity-correction methods introduce artificial boundary conditions which can induce numerical boundary layers, preventing the schemes from converging at optimal rates.

[2]Evans and Hughes prove an optimal convergence rate of $k+1$ for the velocity; for the pressure, they prove a rate of at least k. Their experiments show $k+1$ rates for both variables.

Consistent splitting schemes are schemes that split the time step in a way that does not introduce such artificial boundary conditions. Guermond et al. cite numerical evidence indicating that the velocity- and pressure-correction schemes can have order 3 convergence for velocity and order $\frac{5}{2}$ convergence for pressure (measured in the L^2 norm), for time steps that are not too small relative to the spatial discretization.

Guermond and Minev have recently introduced a new dimensional splitting approach to solving the Navier-Stokes equations [54]. Noting that the Chorin-Temam method can be understood as solving a singular perturbation of the exact equations where the perturbation takes the form $-\epsilon\Delta p$, where ϵ is the size of the time step, they show that the Chorin-Temam scheme is one of a broad class of schemes that have similar convergence properties. Another such scheme is the one they present in the paper, which has the advantage of being extremely cheap to compute: remarkably, it only requires solving a series of one-dimensional boundary-value problems. The approach also is suitable for parallel implementation; in their experiments, they show that their implementation has a speedup close to the ideal one on up to 1000 processors. As presented, the scheme only applies to simple domains discretized into axis-aligned parallelepipeds, although the authors do note in the conclusion several ideas for extending the scheme to more complex domains.

Cockburn et al. have applied their LDG approach to the Navier-Stokes equations [29, 31]. They note that while for many schemes for solving the Stokes and Oseen equations, weakly enforced incompressibility suffices in

the proof of stability, the presence of the nonlinear convective term in the Navier-Stokes equations means that weak enforcement is insufficient. They further note that the standard solution to this problem (due to Temam), which involves a modification of the nonlinearity of the equations, cannot be used in an LDG method because it cannot be rendered in divergence form and therefore would prevent local conservation. The remedy they propose involves projecting the previous approximate solution for the velocity $\boldsymbol{u}_h$ into a divergence-free space; they then use this divergence-free velocity $\boldsymbol{w}$ as the convective velocity in the nonlinear term. They thereby recover the previously studied Oseen problem, so that with the addition of an appropriate stabilization function, the stability of the method is guaranteed. The result is a method that is locally conservative, stable, and converges at optimal rates for polynomial discretizations of arbitrary order. They perform numerical experiments using a classical analytical solution due to Kovasznay [60]. They report success in numerical experiments solving for an analytical solution with Reynolds numbers between 1 and 100; for Reynolds numbers of 1000, they report that the nonlinear iteration does not converge, and hypothesize that this is due to instability in the stationary problem for such Reynolds numbers.

Evans and Hughes have likewise applied their divergence-free B-splines to the steady and unsteady Navier-Stokes equations [46, 47], again with excellent results. They have successfully applied steady Navier-Stokes to the driven cavity problem at Reynolds numbers up to 1000; they report success in other steady flow computations with Reynolds numbers upwards

of 3200. They argue that the fact that most numerical methods only satisfy the incompressibility constraint approximately means that these do not obey fundamental laws of physics, including energy conservation, the failure of which can be shown to lead to numerical instability. By contrast, they demonstrate that their discretizations conserve momentum and satisfy balance laws for energy, vorticity, enstrophy, and helicity.

1.3 Three Popular FEM Codes: deal.II, libMesh, and FEniCS

In this section, we examine three popular object-oriented libraries for doing finite element computations. We begin with a somewhat extended discussion of `deal.II` [4], because it is similar to our own approach in Camellia (see Section 1.5) and because it preceded libMesh and FEniCS. We then highlight some features of libMesh [59] and FEniCS [64].

`deal.II`. `deal.II` is designed for flexibility, ease of use, efficiency, and safety. It is flexible in that it is possible, without too much effort, to vary choices for finite element spaces, spatial dimension, variational formulations, and linear solvers. The ease of use comes largely by virtue of encapsulation: details of complex data storage structures are hidden from the user. Safety is provided by means of runtime parameter checking, which allows programming errors to be detected early. `deal.II` also boasts extensive documentation (5000 pages if printed, they claim), with many implementation examples available on the

`deal.II` website.[3]

It is worth noting that `deal.II` is quite *mature*; the initial DEAL code was developed between 1993 and 1997; `deal.II` was conceived as a rewrite, begun in 1997, and it has continued to be used, extended, and maintained since then. The LDG work cited in Section 1.2 ([27, 30, 29, 31]), for example, uses `deal.II` for its numerical experiments. Also, it is possible to implement DPG methods using `deal.II`; in our research group, Jamie Bramwell has used it to solve inverse problems in elastodynamics.

Several of `deal.II`'s classes are templated on the spatial dimension, making it a relatively simple matter to write dimensionally-independent code. `deal.II` supports *hypercube* topologies in 1, 2, and 3 space dimensions: lines, quads, and hexahedra, with a variety of finite elements supported on these: continuous and discontinuous Lagrange elements, as well as Nédélec and Raviart-Thomas elements. Arbitrary polynomial orders are supported for each of these element types.

`deal.II` allows *h*-, *p*-, and *hp*-adaptive meshes; both anisotropic and isotropic mesh refinements are supported. The meshes are hierarchical: refined (child) cells are nested inside (parent) cells belonging to the previous mesh. Recently, support has been added for distributing `deal.II` meshes using `p4est` [3], a library for parallel adaptive mesh refinement on forests of octrees [21], which boasts scalability to hundreds of thousands of processor cores.

[3] http://www.dealii.org

A consequence of using adaptively refined hierarchical meshes is that refined meshes will usually contain *hanging nodes*—faces where one neighbor has been refined and another has not; see Figure 1.1. If elements K_1, K_2, and K_3 are all linear Lagrangian elements, then along the shared edge, K_2 and K_3 define a space containing all (continuous) functions that are linear along the smaller edges AB and BC, while K_1 only contains functions that are linear along the larger edge AC. `deal.II` addresses this by means of constraints on the spaces for K_2 and K_3, which are incorporated into the final linear problem in a way that will preserve the symmetry and positivity of the stiffness matrix, if the unconstrained stiffness matrix has those features.

Finite element spaces are represented by the `FE` class within `deal.II` — this provides shape functions and their derivatives on the reference cell, when the spaces are defined on a reference cell; otherwise, the functions are provided on physical cells. The `FE` class is, however, not typically used directly by application developers; noting that most finite element developers will only be interested in the *values* of shape functions and their derivatives at particular points (e.g. quadrature points), `deal.II` provides an interface to such values in the `FEValues` class. This class handles the transformation of values from reference to physical space, such as the Piola transforms.[4]

[4]Camellia provides this functionality, among other things, within its `BasisCache` class. See Section 1.5.

libMesh. libMesh [59] is another finite element library; it was in fact inspired by `deal.II`; as the name suggests, libMesh pays particular attention to the mesh—the `Element` class is designed to be subclassed with new elements, and a wide variety of elements are provided. In 2D, both triangle and quad elements are provided; in 3D, hexahedra, tetrahedra, prisms, and pyramids are provided, as well as a collection of infinite elements. *hp*-adaptivity is supported for all element types. Support for distributed stiffness matrix computation and assembly is provided. At present, a copy of the data structures defining the mesh must be stored on each node; development of a `ParallelMesh` class is underway, which will support distributed mesh storage.[5]

FEniCS. The FEniCS project [64] aims at highly automated solution of finite element problems; the authors emphasize the simplicity with which finite element solvers can be implemented. The only topologies currently supported are simplices: intervals in 1D, triangles in 2D, and tetrahedra in 3D. On these topologies, many standard finite element types are supported—most of these support polynomials of arbitrary order. The DOLFIN library [64, Chapter 10] provides the main user interface to FEniCS; this generates C++ code from variational forms specified by the user in the Unified Form Language (UFL) [64, Chapter 17]—the developers aim thereby to achieve simplicity of specification while maintaining performance. Both C++ and

[5]See `http://libmesh.sourceforge.net/doxygen/classlibMesh_1_1ParallelMesh.php`.

Python interfaces are supported; some features are only available using the Python interface.

1.4 DPG for Incompressible Flow

The Discontinuous Petrov-Galerkin Method with Optimal Test Functions. We begin with a short historical review of the method. By a discontinuous Galerkin (DG) method, we mean one that allows test and/or trial functions that are not globally conforming; by a Petrov-Galerkin method, we mean one that allows the test and trial spaces to differ. In 2002, Bottasso et al. introduced a method [10, 11], also called DPG. Like our DPG method, theirs used an "ultra-weak" variational formulation (moving all derivatives to test functions) and replaced the numerical fluxes used in DG methods to "glue" the elements together with new independent unknowns defined on element interfaces. The idea of optimal testing was introduced by Demkowicz and Gopalakrishnan in 2009 [35], which is distinguished by an on-the-fly computation of an approximation to a set of test functions that are optimal in the sense that they realize the supremum in the inf-sup condition. These can then be shown to guarantee minimization of the residual in the dual norm. In 2009-2010, a flurry of numerical experimentation followed, including applications to convection-dominated diffusion [37], wave propagation [78], elasticity [13], thin-body (beam and shell) problems [68], and the Stokes problem [74]. The wave propagation paper also introduced the concept of an *optimal test norm*, whose selection makes the energy norm identical to the

norm of interest on the trial space. In 2010, Demkowicz and Gopalakrishnan proved the convergence of the method for the Laplace equation [36], and Demkowicz and Heuer developed a systematic approach to the selection of a test space norm for singularly perturbed problems [39]. In 2011, Bui-Thanh et al. [19] developed a unified analysis of DPG problems by means of Friedrichs' systems. Our analysis for the Stokes problem, presented here in Chapter 4, builds on the existence of trace spaces and proceeds along a more classical path, connecting to Banach's theory of closed operators.

Some work has been done on nonlinear problems as well. Very early on, Chan, Demkowicz, and Roberts solved the 1D Burgers and compressible Navier-Stokes equations by applying DPG to the linearized problem [23]. Moro et al. have applied their related HDPG method to the 2D Burgers equation; a key difference in their work is that they apply DPG to the *nonlinear* problem, using optimization techniques to minimize the DPG residual. Very recently, Bui-Thanh and Ghattas developed a PDE-constrained optimization approach to DPG [20]; whereas our present approach applies DPG to the linearized problem, Bui-Thanh and Ghattas's approach unifies the treatment of linear and nonlinear problems, allowing direct application of DPG to nonlinear problems, as well as providing an iterative solution technique for DPG problems generally.[6]

[6]Our present work employs only direct solvers; as with any matrix system, it is possible to use standard iterative solvers to solve the system that arises from DPG, however, we do not in general know how to precondition this matrix well—in the context of the Poisson problem, Barker et al. have demonstrated good results using additive Schwarz preconditioners in a

Most DPG analysis assumes that the optimal test functions are computed exactly, but in practice we must approximate them. Gopalakrishnan and Qiu have shown that for the Laplace equation and linear elasticity, for sufficiently high-order approximations[7] of the test space, optimal h-convergence rates are maintained [51].

We will now briefly derive DPG, motivating it as a minimum residual method. Suppose that U is the trial space, and V the test space (both Hilbert) for a well-posed variational problem $b(u, v) = l(v)$. Writing this in the operator form $Bu = l$, where $B : U \to V'$, we seek to minimize the residual for the discrete space $U_h \subset U$:

$$u_h = \underset{u_h \in U_h}{\arg\min} \frac{1}{2} \|Bu_h - l\|_{V'}^2 .$$

Now, the dual space V' is not especially easy to work with; we would prefer to work with V itself. Recalling that the Riesz operator $R_V : V \to V'$ defined by

$$\langle R_V v, \delta v \rangle = (v, \delta v)_V, \quad \forall \delta v \in V,$$

where $\langle \cdot, \cdot \rangle$ denotes the duality pairing between V' and V, is an *isometry*—that is, $\|R_V v\|_{V'} = \|v\|_V$—we can rewrite the term we want to minimize as a norm

conjugate gradient method, but this is to our knowledge the only problem that has been studied thus.

[7] $k_{\text{test}} = k_{\text{trial}} + N$, where N is the number of space dimensions, by k_{test} we mean the polynomial order of the basis functions for the test space, and by k_{trial} we mean the order for the L^2 bases in the trial space.

in V:

$$\frac{1}{2}\left\|Bu_h - l\right\|_{V'}^2 = \frac{1}{2}\left\|R_V^{-1}\left(Bu_h - l\right)\right\|_V^2 = \frac{1}{2}\left(R_V^{-1}\left(Bu_h - l\right), R_V^{-1}\left(Bu_h - l\right)\right)_V. \tag{1.4.1}$$

The first-order optimality condition requires that the Gâteaux derivative of (1.4.1) be equal to zero for minimizer u_h; we have

$$\left(R_V^{-1}\left(Bu_h - l\right), R_V^{-1}B\delta u_h\right)_V = 0, \quad \forall \delta u_h \in U_h.$$

By the definition of R_V, the preceding equation is equivalent to

$$\langle Bu_h - l, R_V^{-1}B\delta u_h\rangle = 0 \quad \forall \delta u_h \in U_h. \tag{1.4.2}$$

Now, if we identify $v_{\delta u_h} = R_V^{-1}B\delta u_h$ as a test function, we can rewrite (1.4.2) as

$$b(u_h, v_{\delta u_h}) = l(v_{\delta u_h}).$$

Note that the last equation is exactly the original variational form, tested with a special function $v_{\delta u_h}$ that corresponds to $\delta u_h \in U_h$; we call $v_{\delta u_h}$ an *optimal test function*. The DPG method is then to solve the problem $b(u_h, v_{\delta u_h}) = l(v_{\delta u_h})$ with optimal test functions $v_{\delta u_h} \in V$ that solve the problem

$$(v_{\delta u_h}, \delta v)_V = \langle R_V v_{\delta u_h}, \delta v\rangle = \langle B\delta u_h, \delta v\rangle = b(\delta u_h, \delta v), \quad \forall \delta v \in V. \tag{1.4.3}$$

In standard conforming methods, test functions are continuous over the entire domain, which would mean that solving (1.4.3) would require computations on the global mesh, making the method impractical. In DPG, we use test

functions that are discontinuous across elements, so that (1.4.3) becomes a local problem—that is, it can be solved element-by-element. Of course, (1.4.3) still requires inversion of the infinite-dimensional Riesz map, and we approximate this by using an "enriched" test space V_h of polynomial order higher than that of the trial space U_h. Note that the test functions $v_{\delta u_h}$ immediately give rise to a hermitian positive definite stiffness matrix; if $\{e_i\}$ is a basis for U_h, we have:

$$b(e_i, v_{e_j}) = (v_{e_i}, v_{e_j})_V = \overline{(v_{e_j}, v_{e_i})}_V = \overline{b(e_j, v_{e_i})}.$$

It should be pointed out that we have not made any assumptions about the inner product on V. An important point is that by an appropriate choice of test space inner product, the induced energy norm on the trial space can be made to coincide with the norm of interest [78]; DPG then delivers the best approximation error in that norm. In practice this optimal test space inner product is approximated by a "localizable" inner product, and DPG delivers the best approximation error up to a mesh-independent constant. That is,

$$\|u - u_h\|_U \leq \frac{M}{\gamma_{DPG}} \inf_{w_h \in U_h} \|u - w_h\|_U ,$$

where $M = O(1)$ and γ_{DPG} is mesh-independent, and γ_{DPG} is of the order of inf-sup constants for the strong operator and its adjoint (see Chapter 4). We therefore say that DPG is *automatically stable*, modulo any error in solving for the test functions $v_{\delta u_h}$.

It is a relatively simple matter, when desired, to enforce *local conservation*—that is, an element-wise property that corresponds to a (mass)

conservation law—by means of Lagrange multipliers. This was first noted by Moro et al. [67]. This is often useful in the context of practical fluid problems. We have not yet, however, explored combining DPG with local conservation in any great detail. In the numerical examples presented in Chapters 6-8, local conservation was not enforced, but it appears from our limited experimentation the context of the Stokes examples to have negligible effect.

Adaptive Strategy for Linear Problems. While other methods typically employ a posteriori error *estimators*, DPG allows the error to be computed precisely. This makes for a simpler, more efficient and more robust adaptivity strategy. When refining a mesh, the key question is where in the mesh the greatest error lies. DPG provides a precise measurement of the error in the dual norm:

$$\|Bu_h - l\|_{V'} = \left\|R_V^{-1}(Bu_h - l)\right\|_V .$$

If we then define an error representation function $e = R_V^{-1}(Bu_h - l) \in V$, we can solve

$$(e, \delta v)_V = b(u_h, \delta v) - l(\delta v), \quad \forall \delta v \in V,$$

locally for e. We use the element contributions to the global residual to drive adaptive mesh refinements.

We use a standard greedy h-refinement strategy:

1. Loop through the elements, determining the maximum element error $\|e_{K\max}\|_V$.

2. Refine all elements with error at least 20% of the maximum $\|e_{K\max}\|_V$.

The analysis and our software implementations support p- and hp-refinements as well. p-adaptivity can employ a strategy similar to the above. We do not yet have a well-defined strategy for deciding between h- and p-refinements once we have decided that a given element should be refined, but in Chapter 6 we include some results for Stokes flow with the h-refinement scheme above as well as ad hoc hp-schemes.

Adaptive Strategies for Nonlinear Problems. When using an adaptive mesh for a nonlinear problem, we have two loops: the refinement loop and the nonlinear iteration. This gives us two basic choices. We can either hold the mesh fixed and iterate through Newton-Raphson steps, or we can first resolve the linearized problem through mesh refinement and only then proceed to the next Newton-Raphson step. One can also imagine hybrid choices, in which the requirement for resolution of the linear problem is more relaxed for early Newton-Raphson steps, becoming stricter when the nonlinear problem is nearly resolved.

Some early experiments led us to the conclusion that refining outside the nonlinear iteration is computationally cheaper. We did not have support for mesh coarsening in the code we were using at the time; having such support could change the balance. We do not currently have mesh coarsening in Camellia, but we do not believe it will be difficult to add, and it may prove

quite useful for nonlinear problems—therefore, we expect to add it in the near future.

Strategies for Transient Problems. Ultimately, we would like to apply DPG to the transient Navier-Stokes equations. When we do so, we expect to start by following Chan et al. in employing space-time finite elements [24], which have the advantage that the entire DPG apparatus can be brought to bear on the problem of appropriate discretization of the time domain while maintaining relatively low memory cost—in this approach, only the degrees of freedom for the current "time slab" are stored. Later, we hope to use *parareal-in-time* elements [63] with DPG, allowing parallel computation in the time domain, through a multi-scale technology. With its precise definition of the error, DPG is ideally suited to automatic adaptivity in the time domain. Such distribution of computations in the time domain will likely be necessary as we move toward exascale computing.

1.5 DPG Software Design

In this section, we describe *Camellia*, a C++ toolbox for rapid development of DPG solvers, implemented atop Sandia's Trilinos library of packages [56]. The essential design goal for Camellia is to make DPG research and experimentation as simple as possible, without sacrificing too much by way of performance. Our ideal is to write code that expresses the DPG formulation—the variational form and the inner product on the test space—in

a way that makes the mathematics transparent, and requires minimal overhead beyond this to specify and solve problems. We aim at excellent software design, following software engineering principles such as encapsulation and using tests and parameter checking to verify the code on an ongoing basis.

The goal of the present section is to provide a broad overview of Camellia, putting it in context by comparison to `deal.II`. Further details can be found in Chapter 3. We begin with some general comments, and then focus, by way of example, on three specific design details: the use of the Factory design pattern throughout the code, how basis values are treated in Camellia, and how hanging nodes are handled.

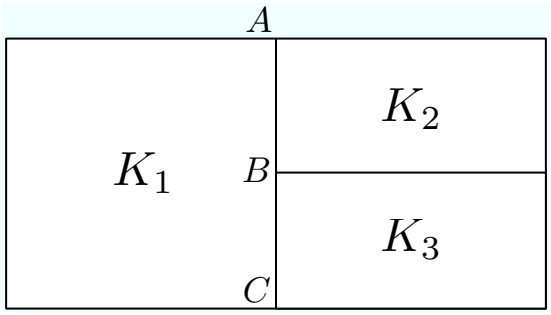


Figure 1.1: Schematic of a hanging node: the element on the right has been refined, resulting in a "broken" edge along the right side of the coarse element K_1.

Because we focus on implementing DPG solvers, we can make certain simplifying assumptions. We can assume a first-order system with all derivatives moved onto test functions. We can assume that trial space variables defined on element interiors are discontinuous across element boundaries; the inter-element "connectivity" is limited to fluxes and traces, which are defined

only on the element boundaries. The latter allows some flexibility in dealing with hanging nodes.

Camellia aims at a higher level of abstraction than does `deal.II`; for instance, Camellia does not require the user to write a loop over the elements in the mesh to compute the stiffness matrix. In fact, the user need not be directly aware of the stiffness matrix at all. There is a tradeoff here: while this has obvious advantages in that the code is generally simpler and easier to write, if the user wishes to do something special with the stiffness matrix, she might find that it is not as obvious how to do so in Camellia as it is in `deal.II`. In Camellia, specification of variational forms and test space inner products hews quite closely to the mathematics; for instance, the code snippet below specifies:

- a field trial variable $u \in L^2$,
- a trace trial variable $\widehat{u} \in H^{1/2}$,
- a test variable $\boldsymbol{v} \in H(\mathrm{div})$,
- a bilinear form $b(u, v) = -\int_\Omega u \nabla \cdot \boldsymbol{v} + \int_{\partial\Omega} \widehat{u} \boldsymbol{v} \cdot \boldsymbol{n}$, and
- a test space norm $\|\boldsymbol{v}\|_V^2 = \|\boldsymbol{v}\|_{L^2}^2 + \|\nabla \cdot \boldsymbol{v}\|_{L^2}^2$.

```
VarFactory varFactory;
// define field variable u
VarPtr u = varFactory.fieldVar("u");
// define flux variable u_hat
VarPtr u_hat = varFactory.traceVar("\\widehat{u}");
// define test function v
```

```
VarPtr v = varFactory.testVar("v", HDIV);
// create bilinear form
BFPtr bilinearForm = Teuchos::rcp( new BF(varFactory) );
// specify field variable term
bilinearForm->addTerm(-u, v->div());
// specify flux variable term
bilinearForm->addTerm(u_hat, v->dot_normal());
// create test space inner product
IPPtr innerProduct = Teuchos::rcp(new IP);
// add L^2 term
innerProduct->addTerm(v);
// add divergence term
innerProduct->addTerm(v->div());
```

Our approach in Camellia can perhaps be seen as falling somewhere between `deal.II` and FEniCS—the specification of variational forms is at least vaguely reminiscent of FEniCS's UFL, but we are not going so far as to provide a domain-specific language and C++ code generation for that language. We do include some basic operator overloading to allow expressions in variational forms such as `sinx * u1 + cosy * u2`, where `sinx` and `cosy` are user-supplied pointers to a `Function` class instance.

At present, Camellia supports only two spatial dimensions, and has no direct support for a time domain. We do hope to add support for one, two, and three space dimensions, as well as extrusion of any of these in a time dimension to create space-time elements. In 2D, Camellia supports both quads and triangles; our hope is to support at least hexahedra and tetrahedra (and perhaps pyramids and prisms) in 3D. This is in contrast to `deal.II`, which for the sake of simplicity of the code supports only hypercube elements, and FEniCS, which supports only simplicial elements.

One of the great advantages of DPG is that we can *measure*, not merely estimate, the error in the dual norm $\|\cdot\|_{V'}$, which is exactly the natural norm from a mathematical point of view (of course it is up to the analyst to specify $\|\cdot\|_V$ well). We can use this to drive adaptivity, and because this depends simply on the choice of norm $\|\cdot\|_V$, the user need not implement an error indicator. At present, we support both h- and p-adaptivity; hp-refinements are also possible, although we do not yet have a general strategy for deciding between h and p. Similarly, the infrastructure for refinements could easily be extended to support anisotropic refinements, but we do not yet have a strategy for deciding the best refinement direction. A host of examples using adaptivity are given in Chapters 6 and 8.

Camellia supports distributed computation of the stiffness matrix; because the computation of optimal test functions is a local problem, this scales perfectly. We do not yet support distributed storage of the mesh or solution—a copy of each is stored on each processor—although we do hope to add both. Currently, we use KLU and MUMPS solvers, which are both direct solvers; MUMPS is a parallel solver. Camellia implements an abstract `Solver` interface, by which users can add linear solvers of their own choosing. (Determining good preconditioners for DPG stiffness matrices is an area of active research.) Detailed discussion of the distribution of the stiffness matrix and some measurements of the scalability of various parts of the code can be found in Appendix A.

The Factory Design Pattern. Camellia represents each basis function that corresponds to a trial or test space variable on a given cell using an instance of Intrepid's `Basis` class. If each cell in the mesh were to store a `Basis` object for each basis, that would be a waste both of memory and the time required to construct the objects. For this reason, we make use of the *Factory* design pattern, which encapsulates object construction—in this instance, we want only to create each basis once on each compute node (to be precise, this is the FlyWeightFactory design pattern [49, p. 199]). When the `BasisFactory` receives a request for a basis belonging to a given function space (e.g. $H(\text{div})$) on a given cell topology (e.g. a triangle) with a given polynomial order (e.g. 5), it looks up this combination of features in a hash map; if such a basis already exists, a pointer to it is returned; if it does not exist, it is created and stored, and a pointer to it is returned.

Discrete spaces in Camellia are represented by the `DofOrdering` class,[8] and the combination of trial space and test space for an element comprise the `ElementType`. Just as with `Basis`, the Factory pattern is applied in `DofOrderingFactory` and `ElementTypeFactory`.

It is worth noting that the Factory classes will not generally be used directly by users. Creation of `DofOrdering`s and `ElementType`s is generally handled by the `Mesh` class; `Basis` instances are usually requested by the `DofOrderingFactory` and assigned to the `DofOrdering`.

[8] We are considering renaming this class `DiscreteSpace`.

Basis Values: `deal.II`'s `FEValues` compared with Camellia's `BasisCache`. Finite element computations often make use of a *reference* cell; values of shape functions and derivatives are computed on the reference cell and transformed to appropriate values in physical space. The values of interest are usually at specific points; most commonly these are quadrature points. Both `deal.II` and Camellia include classes—`FEValues` and `BasisCache`,[9] respectively—that take advantage of this fact to reduce development effort and speed execution.

Both classes compute values in reference space and provide transformations into physical space. Now, not all the basis function values will be of interest for a given computation—derivative values might not be required, for example, and it would be wasteful to compute them in such a case. `deal.II` allows the user to specify that such values are not required so that they will not be computed. Camellia's approach is *lazy* computation of all values: the first time, for example, the derivative values for a basis are requested, these will be computed at all the points of interest, and stored.

It is also worth noting that, while `deal.II`'s `FEValues` class is tied to a particular element type, `BasisCache` is a bit more flexible. `BasisCache` will return values for any requested basis; it is possible to do so efficiently because all bases are created using the `BasisFactory`, as described above.

[9] Because `BasisCache` has evolved somewhat to include additional features relating to the computational context—e.g., it can optionally store a list of the cell IDs currently being operated on—we are considering renaming it `ContextCache`.

`BasisCache` uses the memory address of the `Basis` object as an index into its lookup tables. Generally speaking, a `BasisCache` instance will be relatively short-lived in Camellia, but many methods do take a `BasisCache` pointer as an argument, so that basis values computed in one part of the code might easily be reused in a completely separate part of the code.

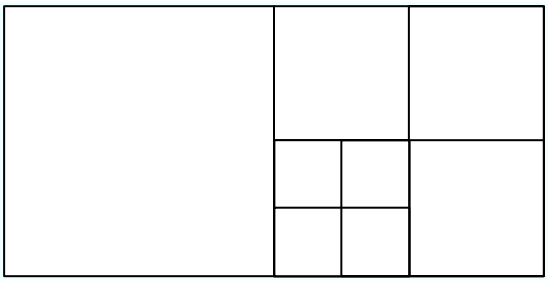


Figure 1.2: Schematic of a 2-irregular mesh.

Treatment of Hanging Nodes: `MultiBasis`. As mentioned in the discussion of `deal.II` in Section 1.3, one of the usual complexities of finite element methods has to do with the treatment of *hanging nodes*, element faces in the mesh where an element has been refined and its neighbor along that face has not, as shown in Figure 1.1. At issue is the relationship of the shape functions discretizing a variable on the coarse element to those discretizing the same variable on the finer neighboring elements. (When neighboring elements are of the same coarseness and polynomial order, the relationship between shape functions on the neighbors is a simple identification.)

The usual way that hanging nodes are handled in finite elements is by imposing a constraint on the finer elements, so that their shape

functions conform to the function space on the coarse element along the shared face. Such constraints might become complicated to implement if arbitrary refinement patterns are allowed—for this reason, finite element codes sometimes limit the irregularity of the mesh;[10] e.g., the mesh shown in Figure 1.2 is called *2-irregular*, because the coarse neighbor on the right has been refined twice along the shared edge, while the one on the left has not been refined at all. The usual practice is to enforce 1-irregularity by introducing additional refinements to coarse neighbors; usually this is implemented by means of constraints imposed on the appropriate degrees of freedom to ensure continuity.

Our approach in Camellia differs. We observe that in DPG the discretizations on element interiors (the L^2 field variables) are entirely independent of each other, so that only the flux and trace variables—the variables defined along a shared edge—need to be reconciled. We therefore allow the coarse element (K_1 in Figure 1.1) to "borrow" the appropriate basis from its finer neighbors.[11] The result is that the coarse element has a basis that is only piecewise polynomial along the shared edge; in the case of fluxes, the basis also allows discontinuity at the hanging node.

To implement this, we define a general basis class, which we call `MultiBasis`, which allows a basis to be formed along a broken edge from

[10] `deal.II` and libMesh both support meshes of arbitrary irregularity, although this is not the default option, and their mechanisms for implementing this necessarily differ from ours.

[11] This is the DPG version of the maximum rule from classical finite elements.

two arbitrary sub-bases. Clearly, to avoid introducing quadrature error, the quadrature points for the `MultiBasis` must be the union of those for its sub-bases; therefore, `MultiBasis` provides these to the `BasisCache` when the latter determines quadrature points for an edge. Because a sub-basis can itself be a `MultiBasis`, we have an elegant recursive mechanism by which we can handle meshes of arbitrary irregularity.[12]

Dual to the concept of `MultiBasis` is that of a `PatchBasis`, in which the finer elements "borrow" a patch of the coarse neighbor's basis.[13] However, here the continuity requirements on traces complicate the picture somewhat—the finer elements impose the constraint that trace values on incident edges agree at the hanging nodes. In the case of `MultiBasis`, this constraint is easily handled by identification of vertex degrees of freedom; in the case of `PatchBasis`, the coarse element's basis has no vertex degree of freedom at the hanging node, so we must add a mechanism for constraining trace bases on the incident edges at hanging nodes.[14]

1.6 Contributions of the Dissertation

The central contribution of the dissertation is the design and development of mathematical techniques and software, based on the DPG method,

[12]Thus far in practice, we have usually continued to enforce 1-irregularity, but in the instances where we have not enforced it, we have not found any ill effects.

[13]This is the DPG version of the minimum rule from classical finite elements.

[14]This is precisely what is missing at present from our implementation of `PatchBasis`. We have tested our implementation, verifying that it works for discretizations that involve only fluxes.

for solving the steady 2D incompressible Navier-Stokes equations in the laminar regime (Reynolds numbers up to about 1000). Along the way, we investigate approximations to these equations—the Stokes equations and the Oseen equations—followed by the steady-state Navier-Stokes equations.

The study of DPG applied to a system of PDEs has several aspects. A first-order variational formulation must be derived, including the selection of appropriate test and trial space norms. The well-posedness of the formulation might be proved. The formulation can be investigated numerically by means of test problems: convergence can be studied by means of manufactured solutions, and "real-world" performance can be studied by means of more physically realistic test problems.

Our mathematical and analytical contributions include the following. We pose several DPG formulations of the Stokes equations—the velocity-gradient-pressure (VGP), velocity-stress-pressure (VSP), and velocity-vorticity-pressure (VVP) formulations. We also pose DPG formulations of the Oseen equations and the 2D incompressible Navier-Stokes equations. We prove the well-posedness of the VGP Stokes formulation for DPG, which has as consequence a guarantee of optimal convergence rates [72].

The core of our contributions to scientific computation is the design and development of a software toolbox (*Camellia*) for the investigation of DPG problems. This supports 2D meshes of triangles and quads of variable polynomial order with possibly curvilinear geometry, provides mechanisms for easy specification of DPG variational forms, support h- and p- refinements,

and supports distributed computation of the stiffness matrix, among other features. We also make a first foray into questions of matrix conditioning and DPG, and suggest some ideas for addressing these.

This dissertation's contribution to mathematical modeling and applications includes, the demonstration of DPG's effectiveness for incompressible flow problems by means of a variety of model problems and manufactured solutions, and the verification that expected convergence rates are achieved and that adaptivity can be used to effectively determine solution features on relatively coarse meshes.

Chapter 2

DPG Step by Step

The purpose of this chapter is to describe in some detail the mathematical and computational steps that must be taken to solve a PDE using DPG. The following chapter is a companion to it, describing the analogous steps required to implement the solver using Camellia. We first go through the steps required to solve the Stokes equations using DPG on a fixed mesh with straight edges, then consider adaptivity, nonlinear equations, and meshes with curvilinear boundaries. We conclude with two alternative Stokes formulations, the velocity-stress-pressure and velocity-vorticity-pressure formulations.

2.1 Stokes on a Fixed Mesh with Straight Edges

We begin with the classical strong form of the Stokes equations:

$$-\nabla p + \mu \Delta \boldsymbol{u} = \boldsymbol{f}$$
$$\nabla \cdot \boldsymbol{u} = 0,$$

on some domain Ω, where $\boldsymbol{u}$ is the velocity, p is the pressure, and $\boldsymbol{f}$ is a vector-valued forcing function. By appropriate non-dimensionalization, we may take $\mu = 1$ without loss of generality. We assume that Dirichlet boundary

conditions $\boldsymbol{u} = \boldsymbol{g}$ are specified on the boundary $\partial\Omega$. We also assume a zero-mean condition on the pressure:

$$\int_\Omega p = 0.$$

The steps to solve the system using DPG are as follows:

1. Determine the variational formulation.
2. Specify boundary conditions.
3. Specify the test space inner product.
4. Define discrete trial and test spaces (including the mesh).
5. Compute optimal test functions.
6. Assemble the stiffness matrix.
7. Solve the global problem.

We treat each of these in turn.

2.1.1 Stokes Variational Formulation

In DPG, our standard practice is to use an *ultra-weak* variational formulation, a first order system in which all derivatives have been moved onto test functions. Generally speaking, there are several ways of doing this. Here, we derive the velocity-gradient-pressure (VGP) formulation for

the Stokes equations. We begin by introducing $\boldsymbol{\sigma} = \nabla \boldsymbol{u}$ to get a first-order system:

$$\begin{aligned} -\nabla p + \nabla \cdot \boldsymbol{\sigma} &= \boldsymbol{f}, \\ \boldsymbol{\sigma} - \nabla \boldsymbol{u} &= 0, \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned}$$

Testing with $(\boldsymbol{v}, q, \boldsymbol{\tau})$, and integrating by parts on a mesh Ω_h with skeleton Γ_h, we have

$$\begin{aligned} (\boldsymbol{\sigma} - p\boldsymbol{I}, \nabla \boldsymbol{v})_{\Omega_h} - \langle (\boldsymbol{\sigma} - p\boldsymbol{I})\boldsymbol{n}, \boldsymbol{v} \rangle_{\Gamma_h} &= (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h}, \\ (\boldsymbol{u}, \nabla q)_{\Omega_h} - \langle \boldsymbol{u} \cdot \boldsymbol{n}, q \rangle_{\Gamma_h} &= 0, \\ (\boldsymbol{\sigma}, \boldsymbol{\tau})_{\Omega_h} + (\boldsymbol{u}, \nabla \cdot \boldsymbol{\tau})_{\Omega_h} - \langle \boldsymbol{u}, \boldsymbol{\tau}\boldsymbol{n} \rangle_{\Gamma_h} &= \boldsymbol{0}. \end{aligned}$$

In the ultra-weak formulation, since we take no derivatives of our trial variables, we take our so-called *field* variables to be in $L^2(\Omega_h)$, which means that we cannot speak of these on the mesh skeleton Γ_h. We therefore introduce new variables $\widehat{\boldsymbol{t}}_n \stackrel{\text{def}}{=} (\boldsymbol{\sigma} - p\boldsymbol{I})\boldsymbol{n}$ and $\widehat{\boldsymbol{u}}$. The hat notation indicates that these variables are only defined on the mesh skeleton—$\widehat{\boldsymbol{t}}_n \in \boldsymbol{H}^{-1/2}(\Gamma_h)$ is a trace of an $H(\text{div})$ function (which we call a *flux* variable), and $\widehat{\boldsymbol{u}} \in \boldsymbol{H}^{1/2}(\Gamma_h)$ is a trace of an $\boldsymbol{H}^1$ function. Defining group variables $u = (\boldsymbol{u}, p, \boldsymbol{\sigma}), \widehat{u} = (\widehat{\boldsymbol{u}}, \widehat{\boldsymbol{t}}_n)$ and $v = (\boldsymbol{v}, q, \boldsymbol{\tau})$, we arrive at our ultra-weak variational formulation:

$$\begin{aligned} b((u, \widehat{u}), v) = (\boldsymbol{\sigma} - p\boldsymbol{I}, \nabla \boldsymbol{v})_{\Omega_h} - \left\langle \widehat{\boldsymbol{t}}_n, \boldsymbol{v} \right\rangle_{\Gamma_h} \\ + (\boldsymbol{u}, \nabla q)_{\Omega_h} - \langle \widehat{\boldsymbol{u}} \cdot \boldsymbol{n}, q \rangle_{\Gamma_h} \\ + (\boldsymbol{\sigma}, \boldsymbol{\tau})_{\Omega_h} + (\boldsymbol{u}, \nabla \cdot \boldsymbol{\tau})_{\Omega_h} - \langle \widehat{\boldsymbol{u}}, \boldsymbol{\tau}\boldsymbol{n} \rangle_{\Gamma_h} = (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h} = l(v). \end{aligned} \tag{2.1.1}$$

2.1.2 Stokes Discretization

DPG does not impose any particular constraints on the basis functions used to represent the various components of the discrete solution. However, we do have the guarantee that the error in the solution is minimized in the energy norm, and under certain modest assumptions (see Chapter 4) if we choose the graph norm as above, we have an inequality of the form

$$\begin{aligned} &\left(\|u-u_h\|^2 + \|\widehat{u}-\widehat{u}_h\|^2_{\hat{H}_A(\Gamma_h)}\right)^{1/2} \\ &\leq \frac{M}{\gamma_{DPG}} \inf_{(w_h,\widehat{w}_h)} \left(\|u-w_h\|^2 + \|\widehat{u}-\widehat{w}_h\|^2_{\hat{H}_A(\Gamma_h)}\right)^{1/2}, \end{aligned} \tag{2.1.2}$$

where the group variables u and $\widehat{u}$, defined above, are the exact solution, while u_h and $\widehat{u}_h$ are their discrete solution counterparts, M and γ_{DPG} are mesh-independent constants, and $\|\cdot\|_{\hat{H}_A(\Gamma_h)}$ is the "natural" norm on the traces, the minimum energy extension norm.

Assuming u is sufficiently smooth, for a discrete L^2 space comprised of polynomials of order k, we expect best h-convergence rates of $k+1$; that is, we have

$$\inf_{w_h \in U_h} \|u-w_h\| \leq C_1 h^{k+1}. \tag{2.1.3}$$

for some mesh-independent constant C_1. It can be shown that, for traces $\widehat{w}_h$ whose $H^{-1/2}(\Gamma_h)$ and $H^{1/2}(\Gamma_h)$ components are approximated by polynomials of orders k and $k+1$, respectively,

$$\inf_{\widehat{w}_h \in \hat{H}_A(\Gamma_h)} \|\widehat{u}-\widehat{w}_h\|_{\hat{H}_A(\Gamma_h)} \leq C_2 h^{k+1}$$

for some mesh-independent constant C_2. For details and further references, see [36, pp. 7-8]. Combining this with equations (2.1.2) and (2.1.3), we then have the bound

$$\|u - u_h\| \leq Ch^{k+1}$$

for $C = \min(C_1, C_2)$.

We can also motivate the choice of polynomial orders for the trial space intuitively from the exact sequence. If we define k as the polynomial order of approximation of field variables, because these belong to L^2, it is natural to choose $k+1$ as the H^1 order. The traces of $H^1(K)$ functions ($\widehat{u}_1$ and $\widehat{u}_2$) belong to $H^{1/2}(\partial K)$, a stronger space than $L^2(K)$, so that $k+1$ is a natural order of approximation for these. The traces of $H(\text{div, K})$ functions $\widehat{\boldsymbol{t}}_n$ belong to $H^{-1/2}(\partial K)$, a weaker space than L^2, so that k is a natural order of approximation for these.

Thus, the natural choice for our trial space discretization is a polynomial space such that

$$\begin{aligned}
&\boldsymbol{u} \in \mathbf{P}^k(K), \widehat{\boldsymbol{u}} \in \mathbf{P}^{k+1}(\partial K), \\
&p \in \mathbf{P}^k(K), \\
&\boldsymbol{\sigma} \in \mathbf{P}^k(K), \widehat{\boldsymbol{t}}_n \in \mathbf{P}^k(\partial K),
\end{aligned}$$

for each element K. With this choice of space, we can expect our discrete solution to converge at a rate of $k+1$, provided that the exact solution is smooth.

Note that the fact that $\widehat{\boldsymbol{u}} \in H^{1/2}(\Gamma_h)$ suggests that we should enforce continuity at vertices; while the fact that $\widehat{\boldsymbol{t}}_n \in H^{-1/2}(\Gamma_h)$ suggests that we should allow discontinuities at the vertices.

Finally, we must choose a discretization for our test space. Again we have considerable freedom, but a natural choice is an element-wise basis that conforms to spaces supporting the differential operators taken on the test space. That is, since we take the divergence of $\boldsymbol{\tau}$, a natural choice for $\boldsymbol{\tau}$ is a vector $H(\text{div})$-conforming basis. In terms of polynomial order, we take the maximum $k+1$ order of discretization used in our trial space, and "enrich" it by some amount Δk. In the present work, we use $\Delta k = 2$. The essential tradeoff is between local computational costs (the choice of test space polynomial order only affects the determination of optimal test functions, a local operation) and accurate determination of the optimal test functions.

Thus we select a test space such that

$$\begin{aligned}
&\boldsymbol{v} \in \mathbf{P}^{k+1+\Delta k}(K) \cap \boldsymbol{H}^1(K), \\
&q \in \mathbf{P}^{k+1+\Delta k}(K) \cap H^1(K), \\
&\boldsymbol{\tau} \in \mathbf{P}^{k+1+\Delta k}(K) \cap \boldsymbol{H}(\text{div, K}) .
\end{aligned}$$

The specific choices of test and trial functions available in Camellia will be treated in Chapter 3.

2.1.3 Stokes Boundary Conditions

As indicated above, we take our field variables—that is, $\boldsymbol{u}, p$, and $\boldsymbol{\sigma}$—to be in L^2. We therefore replace the boundary condition $\boldsymbol{u} = \boldsymbol{g}$ on $\partial\Omega_h$ with the condition $\widehat{\boldsymbol{u}} = \boldsymbol{g}$ on $\partial\Omega_h$. We will treat the zero-mean condition on the pressure in Section 2.1.6.

2.1.4 Stokes Test Space Inner Product

The test space inner product is a crucial choice in DPG; it determines the energy norm, in which the method is optimal. Suppose we want a method optimal in the L^2 norm of our field variables; that is, we wish to minimize

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|^2 + \|\boldsymbol{\sigma} - \boldsymbol{\sigma}_h\|^2 + \|p - p_h\|^2 .$$

As we will discuss in more depth in Chapter 4, an appropriate choice for the test norm in this case is the *graph norm*. Because our focus in the present discussion is on execution, here we limit ourselves to the determination of the graph norm. Grouping equation (3.2.1) by the field variables, we have

$$\begin{aligned} &= (\boldsymbol{u}, \nabla q + \nabla \cdot \boldsymbol{\tau})_{\Omega_h} + (p, \nabla \cdot \boldsymbol{v})_{\Omega_h} + (\boldsymbol{\sigma}, \nabla \boldsymbol{v} + \boldsymbol{\tau})_{\Omega_h} \\ &\quad + \langle \text{boundary terms} \rangle \end{aligned}$$

The graph norm is then given by the Euclidean combination of the test terms for each field variable, plus L^2 norms[1] of each test variable:

$$\|(\boldsymbol{v}, q, \boldsymbol{\tau})\|^2_{\text{graph}} = \|\nabla q + \nabla \cdot \boldsymbol{\tau}\|^2 + \|\nabla \cdot \boldsymbol{v}\|^2 + \|\nabla \boldsymbol{v} + \boldsymbol{\tau}\|^2 + \|\boldsymbol{v}\|^2 + \|q\|^2 + \|\boldsymbol{\tau}\|^2 .$$

[1]The L^2 terms may be weighted by constants; the best choice of constants will depend on the problem. For Stokes, we have simply taken unit weights.

2.1.5 Optimal Test Function Determination

As discussed in Chapter 3, each trial space basis function e_i has corresponding to it an optimal test function v_{e_i}, where v_{e_i} solves the equation

$$(v_{e_i}, \delta v)_V = b(e_i, \delta v) \quad \forall \delta v \in V,$$

where V is the (discrete) test space. Because we employ test functions that are allowed to be discontinuous at inter-element boundaries, this is an element-local problem. Note that because of the discretizations we have selected, the test space will have more degrees of freedom—if we take the test space to have m degrees of freedom per element and the trial space to have n degrees of freedom, then the left hand side of this system will be a square $m \times m$ matrix, and the right a rectangular $m \times n$ matrix (expanding the equation column-wise in the trial space index i), and the solution will be n vectors of length m: the m coefficients of each optimal test function. If our basis for the enriched test space consists of functions v_j and we define the $m \times m$ Gram matrix

$$G_{jk} = (v_j, v_k)_V$$

and the $n \times m$ bilinear form matrix

$$B_{ij} = b(e_i, v_j),$$

then the optimal test coefficients are the columns of $G^{-1}B^T$.

2.1.6 Stiffness Matrix Assembly

Once the optimal test functions for an element are computed, the element stiffness matrix K_{ij} is relatively simple to compute; we have

$$K_{ij} = b(e_i, v_{e_j}) = (v_{e_i}, v_{e_j})_V,$$

by the very problem that we solved to determine the optimal test functions. Note that we can compute the inner product of an arbitrary pair of test basis functions $\alpha_i v_i$ and $\beta_j v_j$ by means of the Gram matrix:

$$(\alpha_i v_i, \beta_j v_j)_V = \boldsymbol{\alpha}^T (v_i, v_j)_V \boldsymbol{\beta} = \boldsymbol{\alpha}^T G \boldsymbol{\beta},$$

so that the element stiffness matrix is given by

$$(v_{e_i}, v_{e_j})_V = (G^{-1}B^T)^T G G^{-1} B^T = B G^{-1} B^T.$$

The element stiffness matrix entries can then be assembled into a global stiffness matrix. Note that the only inter-element coupling comes through the abstract trace variables (the trace $\widehat{\boldsymbol{u}}$ and flux $\widehat{\boldsymbol{t}}_n$, in the case of Stokes).

To impose the zero-mean condition on the pressure, we follow a technique described by Bochev and Lehoucq [7], which preserves symmetric positive definiteness of the stiffness matrix and allows a simple implementation for nodal bases (which is what we use for field variables in Camellia).

2.1.7 Global Problem Solution

As noted above, the only entries in our stiffness matrix that are coupled between elements belong to trace variables. We can therefore employ static

condensation to reduce the size of the global system. That is, the matrix can be reordered to take the form

$$\begin{pmatrix} K_{11} & K_{12} \\ K_{12}^T & K_{22} \end{pmatrix} \begin{pmatrix} u \\ f \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$$

where K_{11} is block diagonal, u represents the degrees of freedom corresponding to field variables, and f those corresponding to the trace variables. Noting that $u = K_{11}^{-1}(F_1 - K_{12}f)$, we can substitute this into the equation $K_{12}^T u + K_{22}f = F_2$ to obtain an equation for the trace degrees of freedom:

$$(K_{22} - K_{12}^T K_{11}^{-1} K_{12})f = F_2 - K_{12}^T K_{11}^{-1} F_1.$$

Since K_{11} is block diagonal, its inversion can be carried out element-wise and in parallel; since K_{12} is a significantly smaller matrix, the computational cost of the global solve is reduced. For simplicity, we have generally employed direct solvers for both the local and the global solves.

2.2 Riesz Representations and Adaptivity

DPG minimizes the error in the energy norm; that is, for a problem with exact solution u and discrete solution u_h,

$$\|u - u_h\|_E = \sup_{\|v\|_V = 1} b(u - u_h, v)$$

is minimized. Now, we may rewrite this as

$$\begin{aligned} \sup_{\|v\|_V = 1} b(u - u_h, v) &= \sup_{\|v\|_V = 1} (b(u, v) - b(u_h, v)) \\ &= \sup_{\|v\|_V = 1} (l(v) - b(u_h, v)) \\ &= \|l - Bu_h\|_{V'}, \end{aligned}$$

where the operator B is defined by $Bu = b(u, \cdot)$. The error $\|l - Bu_h\|_{V'}$ is computable by virtue of the Riesz representation theorem, which tells us that there exists a function $e \in V$ such that

$$(e, v)_V = l(v) - b(u_h, v) \quad \forall v \in V,$$

and moreover that $\|e\|_V = \|l - Bu_h\|_{V'}$. We call this e the *error representation function*, and compute it by defining a residual vector

$$r_i = l(v_i) - b(u_h, v_i),$$

then inverting the Gram matrix to solve for $e = G^{-1}r$. We can then compute

$$\|e\|_V^2 = (e, e)_V = e^T G e = (G^{-1}r)^T G e = r^T e.$$

Note that this computation is local to the element, making it susceptible to parallel execution. Once we have computed the error, any number of refinement strategies might be employed. Our basic strategy is a greedy one, determining the maximum element error $\|e_K\|_V$ in the mesh, and marking for refinement any element with error greater than $\theta \|e_K\|_V$, where $\theta \in (0, 1)$ is a threshold parameter, which in most of the computations here presented we have taken to be 0.20. The refinements might be either h- or p-refinements.

When one element is more refined than its neighbor (in either h or p), we must decide what discretization to use for the trace variables along the interface between the elements. While other choices are possible, our usual approach is to adopt the finer element's discretization along the interface.

2.3 Nonlinear Problems

DPG minimizes the residual of a linear problem. While more sophisticated approaches are possible, for the present when solving a nonlinear problem we first linearize the problem, then use DPG to solve the linearized problem. For example, we may write the steady Navier-Stokes equations as:

$$\begin{aligned} -\nabla p + \mu \nabla \cdot \boldsymbol{\sigma} &= \boldsymbol{f} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}, \\ \boldsymbol{\sigma} - \nabla \boldsymbol{u} &= 0, \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned}$$

Observing that the nonlinear, convective term may be written $\boldsymbol{u} \cdot \nabla \boldsymbol{u} = \boldsymbol{u} \cdot \boldsymbol{\sigma}$, we immediately see that a Navier-Stokes formulation corresponding to our VGP Stokes formulation is

$$\begin{aligned} \langle \widehat{\boldsymbol{t}}_{\boldsymbol{n}}, \boldsymbol{v} \rangle_{\Gamma_h} + (p, \nabla \cdot \boldsymbol{v})_{\Omega_h} + (\boldsymbol{\sigma}, \nabla(\mu \boldsymbol{v}))_{\Omega_h} - (\boldsymbol{u} \cdot \boldsymbol{\sigma}, \boldsymbol{v})_{\Omega_h} &= (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h}, \\ (\boldsymbol{\sigma}, \boldsymbol{\tau})_{\Omega_h} - \langle \widehat{\boldsymbol{u}}, \boldsymbol{\tau} \boldsymbol{n} \rangle_{\Gamma_h} + (\boldsymbol{u}, \nabla \cdot \boldsymbol{\tau})_{\Omega_h} &= 0, \\ \langle \widehat{\boldsymbol{u}} \cdot \boldsymbol{n}, q \rangle_{\Gamma_h} - (\boldsymbol{u}, \nabla q)_{\Omega_h} &= 0. \end{aligned}$$

If we define the Stokes bilinear formulation as $b_{\text{Stokes}}(u, v) = l_{\text{Stokes}}(v)$, and linearize about $(\boldsymbol{u} + \Delta \boldsymbol{u}, \boldsymbol{\sigma} + \Delta \boldsymbol{\sigma}, p + \Delta p)$, we then have

$$b_{\text{Stokes}}(\Delta u, v) - (\Delta \boldsymbol{u} \cdot \boldsymbol{\sigma} + \boldsymbol{u} \cdot \Delta \boldsymbol{\sigma}, \boldsymbol{v})_{\Omega_h} = (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h} - b_{\text{Stokes}}(u, v) + (\boldsymbol{u} \cdot \boldsymbol{\sigma}, \boldsymbol{v})_{\Omega_h}.$$

Our strategy for solving this is a standard Newton iteration: we start from some initial guess $u = u_0$, solve for the increment Δu, set $u_{i+1} = u_i + \Delta u$, and iterate until some measure of the increment is below a desired threshold. It is

worth noting, however, that in general our test norm will now depend on the background flow[2] u_i.

2.4 Curvilinear Meshes

When using curved geometries, the order of finite element solution convergence is generally limited by the order of approximation of the geometry. Thus to achieve higher-order convergence we require higher-order geometry representations. We choose to employ an isoparametric representation of the geometry, in which the basis functions used in finite element computations are also used to represent the geometry. Compared with representing the geometry exactly in our computations, this is an appealing choice for two reasons: first, the isoparametric representation will in general be cheaper to compute; second, isoparametric geometry allows the exact representation of *linearized rigid-body motion*, a fact of engineering interest. Our approach here essentially follows Demkowicz et al. [34, pp. 195-210].

Consider a connected domain $\Omega \subset \mathbb{R}^2$. We assume that the domain can be partitioned into curvilinear triangles and quadrilaterals. An example of a domain with such a partition is shown in Figure 2.1. In order to work with such geometries, we must determine a systematic mechanism for constructing maps $\boldsymbol{x}(\boldsymbol{t})$ from reference elements to physical space.

Now, typically an engineer or analyst will not have ready to hand

[2]That is, the previous solution.

a representation of the interior of the domain; rather, there will be some description of the boundary of the domain. In what follows we assume that this description is parametrically defined. This assumption does impose some constraints—in some cases, geometry will be known implicitly, perhaps defined by the intersection of two surfaces. For further details on implicitly defined geometry, see [34, pp. 198-199] and [40, pp. 96-97].

We proceed as follows. Since we assume a parametrization of the domain boundary, it is natural further to assume a parametrization of the edges (interior edges might simply be taken to be straight lines). From the edge parametrizations it is possible to construct a *transfinite interpolant* [52], a mapping from a reference quadrilateral or triangle onto its curvilinear counterpart which is exact on the edges. The transfinite interpolant in hand—a representation of the exact geometry—we construct a *projection-based interpolant*, which approximates the geometry using precisely the polynomial

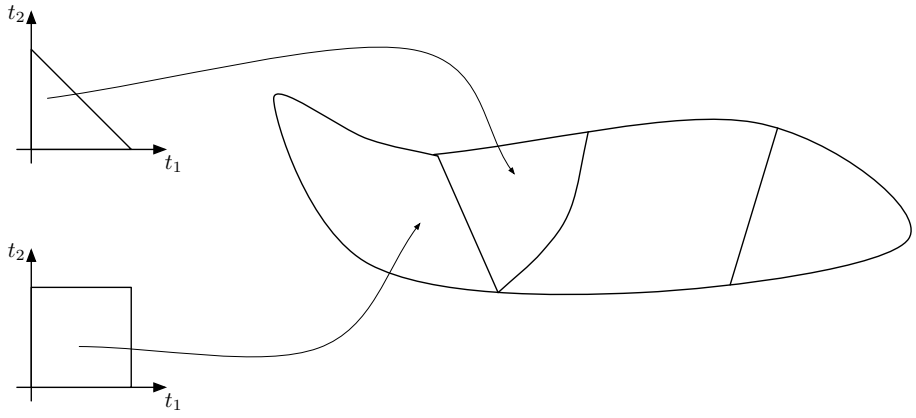


Figure 2.1: An example curvilinear domain partitioned into triangular and quadrilateral elements.

basis employed in our finite element discretization.

Suppose that for an element K, for each edge $e_i \subset \partial K$ connecting vertices $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i+1}$ there is some given parametrization $\boldsymbol{x}_i : [0,1] \to e_i$ onto the edge such that $\boldsymbol{x}_i(0) = \boldsymbol{v}_i$ and $\boldsymbol{x}_i(1) = \boldsymbol{v}_{i+1}$. (We implicitly use modular arithmetic in vertex here and below—thus on a quadrilateral vertex 0 is identified with vertex 4, and similarly 0 and 3 are identified on the triangle.) The vertex and edge numbering for the reference quadrilateral are shown in Figure 2.2. We now construct transfinite interpolants for quadrilateral elements; an analogous construction exists for triangles.

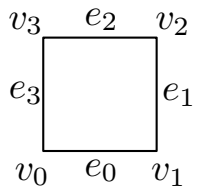


Figure 2.2: Vertex and edge numbering on reference quadrilateral.

Transfinite Interpolation on the Quadrilateral. We define the edge bubble functions[3] $\Delta\boldsymbol{x}_i$ by subtracting the vertex contributions from the edge parametrizations in an appropriately blended fashion:

$$\Delta\boldsymbol{x}_i \stackrel{\text{def}}{=} \boldsymbol{x}_i - (t-1)\,\boldsymbol{v}_i - t\,\boldsymbol{v}_{i+1}.$$

[3]A bubble function on a topological entity is defined to be a function that vanishes on that entity's boundary. Thus the edge bubbles vanish at the vertices, face bubbles vanish on edges, and so on.

We define a bilinear interpolant $\boldsymbol{b}(t_1, t_2)$ determined by the vertices:

$$\boldsymbol{b}(t_1, t_2) \stackrel{\text{def}}{=} (1 - t_1)\,(1 - t_2)\,\boldsymbol{v}_0 + t_1\,(1 - t_2)\,\boldsymbol{v}_1 + t_1\,t_2\,\boldsymbol{v}_2 + (1 - t_1)\,t_2\,\boldsymbol{v}_3.$$

The transfinite interpolant is then given by

$$\begin{aligned}\boldsymbol{x}(t_1, t_2) \stackrel{\text{def}}{=} \boldsymbol{b}(t_1, t_2) &+ (1 - t_2)\,\Delta x_0(t_1) + t_2\,\Delta x_2(t_1) \\ &+ (1 - t_1)\,\Delta x_3(t_2) + t_1 \Delta x_1(t_2).\end{aligned}$$

It is clear from the construction that this agrees exactly with the curves $\boldsymbol{x}_i$ on the element boundary, and that it blends the $\boldsymbol{x}_i$ in a continuous fashion on the element interior.

$\boldsymbol{H}^1$ projection-based interpolation. Thus far, our representation of the geometry has been exact. For reasons suggested above, we would like to determine a polynomial approximation to the geometry which will also allow us to maintain convergence rates. The correct way to do so is known as projection-based interpolation [34, pp. 180-183]. In our finite element discretization, it is desirable that elements should be *compatible* in the sense that they agree exactly on the location of shared vertices (and vertices on the approximate boundary should coincide with the exact boundary) as well as on the approximation of shared edges. Further, we would like our approximate geometry to closely approximate both location (that is, value) and curvature (derivative) information. We thus arrive at a constrained projection problem, which we can divide into steps:

1. Interpolate the vertices (i.e. use the vertex locations to set weights for the vertex basis functions).

2. Project the exact transfinite interpolant bubble—that is, $\Delta\boldsymbol{x} \stackrel{\text{def}}{=} \boldsymbol{x}(t_1, t_2) - \boldsymbol{b}(t_1, t_2)$—into the discrete space of edge bubble functions. Call the sum of the weighted edge and vertex functions $\widetilde{\boldsymbol{x}}_{\text{edge}}$.

3. Project the difference of the function thus far and the transfinite interpolant—that is, $\Delta\boldsymbol{x} - \widetilde{\boldsymbol{x}}_{\text{edge}}$—into the discrete space of face bubble functions.

Refinements. It remains to specify how refinements should be handled: specifically, should the edges interior to the refined element be curves defined as above by the transfinite interpolant, or will straight edges suffice? For standard conforming elements, the usual practice is to use curved edges on the interior to guarantee optimal convergence rates. We were unsure whether this would be necessary for DPG in the curvilinear geometry of immediate interest to us, so we simply experimented with using straight edges on element interiors—the computational advantage being that we reduce the number of elements for which we need to compute the curvilinear geometry, and the refinements are somewhat simpler to implement this way. It is worth noting that we do use the transfinite interpolant to compute the location of the new vertices. As documented in Chapter 5, in a Poisson manufactured solution on a domain with an embedded circle, we match the best approximation in the space using this approach. This is not a proof, and we do hope in the future

to implement refinements that are curvilinear on their interiors so that we can study the relative costs and benefits of the two approaches.

It is worth noting in this context that for certain meshes and geometries, clearly refinements with straight-edged interiors will *not* suffice—for example, if a sufficiently thin, sufficiently curved shell element is refined, the straight edges on the interior of a refinement will intersect the curved edges on the element's boundary. In the present work, we only employ one curvilinear geometry of modest curvature, and this is what we use to verify our approach in Chapter 5.

2.5 Two other Stokes formulations: VVP and VSP

We conclude this chapter with two alternative Stokes formulations. While the focus of this dissertation is on the VGP formulation derived above, this is not the only possibility, and we have also made use of both a velocity-stress-pressure (VSP) formulation as well as a velocity-vorticity-pressure (VVP) formulation, each of which we detail below. As we will discuss in Chapter 5, we have verified that each of these attain the expected convergence rates when used on smooth manufactured solutions.

Stokes VSP Formulation. We begin with the formulation we used in our original work with the Stokes problem, a velocity-stress-pressure (VSP) formulation.

The strong form of the Stokes problem with which we begin is as follows:

$$-2\mu\nabla\cdot\boldsymbol{\epsilon}+\nabla p=\boldsymbol{f} \qquad \text{in } \Omega, \tag{2.5.4}$$

$$\nabla\cdot\boldsymbol{u}=0 \qquad \text{in } \Omega, \tag{2.5.5}$$

$$\boldsymbol{u}=\boldsymbol{g_D} \qquad \text{on } \partial\Omega, \tag{2.5.6}$$

where $\Omega\subset\mathbb{R}^2$, μ is viscosity, $\boldsymbol{\epsilon}=\nabla^{\text{sym}}\boldsymbol{u}$ is strain, p is pressure, u velocity, and $\boldsymbol{f}$ a vector forcing function.

We introduce stress $\boldsymbol{\sigma}$ and (tensor) vorticity $\boldsymbol{\omega}$ by

$$\begin{aligned}\boldsymbol{\sigma}&=2\mu\boldsymbol{\epsilon}-p\boldsymbol{I},\\ \boldsymbol{\omega}&=\frac{1}{2}(\nabla\boldsymbol{u}-\nabla\boldsymbol{u}^T),\end{aligned}$$

so that equation (2.5.4) becomes simply $-\nabla\cdot\boldsymbol{\sigma}=\boldsymbol{f}$. We also have

$$\boldsymbol{\epsilon}=\frac{1}{2\mu}(\boldsymbol{\sigma}+p\boldsymbol{I}).$$

Since $\boldsymbol{\epsilon}=\nabla^{\text{sym}}\boldsymbol{u}=\nabla\boldsymbol{u}-\boldsymbol{\omega}$, the entire system is

$$\begin{aligned}\frac{1}{2\mu}(\boldsymbol{\sigma}+p\boldsymbol{I})-\nabla\boldsymbol{u}+\boldsymbol{\omega}&=\boldsymbol{0} && \text{in } \Omega,\\ -\nabla\cdot\boldsymbol{\sigma}&=\boldsymbol{f} && \text{in } \Omega,\\ \nabla\cdot\boldsymbol{u}&=0 && \text{in } \Omega,\\ \boldsymbol{u}&=\boldsymbol{g_D} && \text{on } \partial\Omega.\end{aligned}$$

Note that the antisymmetric part of the first equation recovers the definition of $\boldsymbol{\omega}$, so that it need not enter the system separately. That is, in 2D the vorticity

reduces to scalar $\omega = \omega_{21} = \frac{1}{2}(u_{1,2} - u_{2,1})$. Our strong formulation is

$$\begin{aligned}
\frac{1}{2\mu}\begin{pmatrix}\sigma_{11}+p\\ \sigma_{21}\end{pmatrix} - \nabla u_1 + \begin{pmatrix}0\\ \omega\end{pmatrix} &= \mathbf{0} && \text{in } \Omega,\\
\frac{1}{2\mu}\begin{pmatrix}\sigma_{12}\\ \sigma_{22}+p\end{pmatrix} - \nabla u_2 - \begin{pmatrix}\omega\\ 0\end{pmatrix} &= \mathbf{0} && \text{in } \Omega,\\
-\nabla \cdot \begin{pmatrix}\sigma_{11}\\ \sigma_{21}\end{pmatrix} &= f_1 && \text{in } \Omega,\\
-\nabla \cdot \begin{pmatrix}\sigma_{12}\\ \sigma_{22}\end{pmatrix} &= f_2 && \text{in } \Omega,\\
\nabla \cdot \boldsymbol{u} &= 0 && \text{in } \Omega,\\
\boldsymbol{u} &= \boldsymbol{g_D} && \text{on } \partial\Omega.
\end{aligned}$$

Multiplying the first two equations by vector test functions $\boldsymbol{q}_i$ and the following three by scalar test functions v_i, and integrating by parts over an element K, we obtain

$$\begin{aligned}
\int_K \left(\frac{1}{2\mu}\begin{pmatrix}\sigma_{11}+p\\ \sigma_{21}\end{pmatrix} + \begin{pmatrix}0\\ \omega\end{pmatrix}\right)\cdot \boldsymbol{q}_1 + \int_K u_1 \nabla\cdot \boldsymbol{q_1} \quad - \int_{\partial K} \widehat{u}_1 \boldsymbol{q_1}\cdot n \quad &= \mathbf{0},\\
\int_K \left(\frac{1}{2\mu}\begin{pmatrix}\sigma_{12}\\ \sigma_{22}+p\end{pmatrix} - \begin{pmatrix}\omega\\ 0\end{pmatrix}\right)\cdot \boldsymbol{q}_2 + \int_K u_2 \nabla\cdot \boldsymbol{q_2} \quad - \int_{\partial K} \widehat{u}_2 \boldsymbol{q_2}\cdot n \quad &= \mathbf{0},\\
\int_K \begin{pmatrix}\sigma_{11}\\ \sigma_{21}\end{pmatrix}\cdot \nabla v_1 \quad - \int_{\partial K} \widehat{\boldsymbol{\sigma}}_{1n} v_1 \quad &= \int_K f_1 v_1,\\
\int_K \begin{pmatrix}\sigma_{12}\\ \sigma_{22}\end{pmatrix}\cdot \nabla v_2 \quad - \int_{\partial K} \widehat{\boldsymbol{\sigma}}_{2n} v_2 \quad &= \int_K f_2 v_2,\\
-\int_K \boldsymbol{u}\cdot \nabla v_3 \quad + \int_{\partial K} \widehat{\boldsymbol{u}} v_3 \cdot n \quad &= 0.
\end{aligned}$$

Stokes Velocity-Vorticity-Pressure Formulation. The standard velocity-vorticity-pressure (VVP) Stokes formulation is:

$$\begin{aligned} \nabla \times \omega + \nabla p &= \boldsymbol{f}, \\ \omega - \nabla \times \boldsymbol{u} &= 0, \\ \nabla \cdot \boldsymbol{u} &= 0. \end{aligned}$$

(Note that the value of ω here differs by a factor of $-\frac{1}{2}$ from the ω as defined in the previous formulation.) Multiplying by test functions, integrating by parts, and substituting fluxes and traces for boundary values, we obtain:

$$\begin{aligned} \int_{\partial K} \widehat{\omega} \boldsymbol{q} \cdot \begin{pmatrix} n_2 \\ -n_1 \end{pmatrix} + \int_{\partial K} \widehat{p} q_n - \int_K \omega \nabla \times \boldsymbol{q} - \int_K p \nabla \cdot \boldsymbol{q} &= \int_K \boldsymbol{f} \cdot \boldsymbol{q}, \\ \int_{\partial K} \widehat{u}_{\times \boldsymbol{n}} v_1 - \int_K \boldsymbol{u} \cdot (\nabla \times v_1) + \int_K \omega v_1 &= 0, \\ \int_{\partial K} \widehat{u}_{\boldsymbol{n}} v_2 - \int_K \boldsymbol{u} \cdot (\nabla v_2) &= 0\,. \end{aligned}$$

Whereas the original Stokes formulation required 7 field variables, 3 traces, and 2 fluxes, the VVP formulation requires just 4 field, 2 trace, and two flux variables.

Chapter 3

Camellia Step by Step

In this chapter, we describe some details of *Camellia*, a C++ toolbox for rapid development of DPG solvers, implemented atop Sandia's Trilinos library of packages [56]. We will broadly follow the outline of Chapter 2, describing how to use Camellia to define and invoke the mathematical and computational apparatus specified there. As there, we begin with the steps required to solve the Stokes equations using DPG on a fixed mesh with straight edges, then consider adaptivity, nonlinear equations, and meshes with curvilinear boundaries.

The essential design goal for Camellia is to make DPG research and experimentation as simple as possible, without sacrificing too much by way of performance. Our ideal is to write code that expresses the DPG formulation—the variational form and the inner product on the test space—in a way that makes the mathematics transparent, and requires minimal overhead beyond this to specify and solve problems. We aim at excellent software design, following software engineering principles such as encapsulation and using tests and parameter checking to verify the code on an ongoing basis.

Key features of Camellia include:

- simple implementation of systems of arbitrary first-order PDEs,
- simple implementation of arbitrary test space inner products,
- distributed stiffness matrix determination,
- distributed solve (using MUMPS),
- support for h-, p-, and hp-adaptivity,
- support for meshes made up of quads and triangles,
- support for meshes of arbitrary irregularity, and
- H(grad)-, $H(\mathrm{div})$-, and $H(\mathrm{curl})$-conforming basis functions.

Reference-counted pointers. Throughout the code, and in this chapter, we make liberal use of reference-counted pointers (RCPs, instances of the `Teuchos::RCP` template class), which allow us to create objects without much concern about reclaiming the memory allocated for them: when the last reference to the object goes out of scope, the memory will automatically be reclaimed. Because `Teuchos::RCP<`*`ClassName`*`>` can be a lot to type, in the code and in this chapter, we adopt a convention that *`ClassName`*`Ptr` will mean the same thing.

3.1 Building Blocks for DPG Formulations: Var, LinearTerm, Function, IP, BF

Here, we introduce a few key classes which provide building blocks to specify a DPG formulation.

`Var`. The `Var` class identifies a trial or test space variable. The `VarFactory` class manages creation of `Var`s, ensuring unique identifiers for each variable, which in turn are used in the ordering of corresponding degree of freedom coefficients in the `DiscreteSpace` class. The `Var` class also keeps track of a linear operator applied to the variables—the default is a value operator; first-order differential operators, operators involving unit normals, and operators to select a component of a vector function are also available. `Var`s also have *types*: field, trace, flux, and test types are defined. Each `Var` has a tensorial *rank*—0 for scalar variables, 1 for vector variables, and so on.

`Function`. The `Function` class defines a function that may vary in space. The `values()` method takes a `BasisCache` (recall from Chapter 1 that `BasisCache` defines the computational context) as an argument; the `BasisCache` in turn defines points of interest. We adopt this approach for several reasons. First, by operating on batches of points, computations may be more efficient than if we required a function call for each point. Second, the `BasisCache` may contain other contextual information (for example, identifiers for the mesh elements being operated on) on which the `Function`

instance depends—compared with adding arguments to the `values()` for such information, this allows a simple, unified interface for all `Function` instances. Finally, as its name suggests, the `BasisCache` caches previously computed basis values, which allows some `Function` instances to compute their values more quickly—for instance, `PreviousSolutionFunction` uses computed solution coefficients to determine solution values, and this requires summing over various basis functions. As with `Var`, `Function` has a rank.

There are many `Function` subclasses provided, and each of these can be used for visualization output; here, we name a few representative examples. The `PreviousSolutionFunction` takes as its argument a `LinearTermPtr` (see below), so that solution variables may be combined in an arbitrary fashion and used elsewhere. This is exactly the mechanism by which nonlinear problems are solved in Camellia. The `hFunction` allows simple definition of functions that depend on the diameter of mesh cells. `MeshPolyOrderFunction` has as its value the polynomial order trial space on each cell of the mesh; this is particularly useful for visualization of variable-order meshes.

`LinearTerm.` The `LinearTerm` class represents variational terms that are linear in either trial or test variables. As such, `LinearTerm`s are defined as the product of a `Function` and a `Var`. `LinearTerm`s have types and ranks determined by their component functions and variables. Operator overloading allows simple syntax for some commonly used patterns—if `f1, f2` are `FunctionPtr`s and `v1, v2` are test variables, for example, each of the

following expressions returns a corresponding `LinearTermPtr`:

- `f1 * v1`,
- `v1 / f1`,
- `f1 / f2 * v1`,
- `v1 + v2`,
- `f1 * v1 + f2 * v2`,
- `3 * v1 + v2 / 5`, and
- `- v1`.

Moreover, application of differential operators to variables is just a matter of calling the appropriate method on the `VarPtr`s; e.g. `v1->grad()` means ∇v_1.

`IP`. The `IP` class represents an inner product; its primary purpose is to define the test space inner product. Its `addTerm()` method takes a `LinearTermPtr` as argument. The square of the norm generated by the inner product is the sum of the squares of its terms. For example, the following code will specify an inner product that generates the norm $\left(\|v\|^2 + \|\nabla v\|^2\right)^{1/2}$.

```
IPPtr ip =  = Teuchos::rcp(new IP());
ip->addTerm(v);
ip->addTerm(v->grad());
```

`BF`. The `BF` class represents a bilinear form; its `addTerm()` method takes as arguments trial and test space `LinearTermPtr`s. Each term is the L^2 inner product of its arguments, taken either on the interior of each element (for field variables) or along the element boundary (for fluxes and traces).

3.2 Stokes on a Fixed Mesh with Straight Edges

3.2.1 Stokes Variational Formulation

Recall the VGP formulation we defined for Stokes in Chapter 2:

$$\begin{aligned} b(u,v) = (\boldsymbol{\sigma} - p\boldsymbol{I}, \nabla \boldsymbol{v})_{\Omega_h} - \left\langle \widehat{\boldsymbol{t}}_n, \boldsymbol{v} \right\rangle_{\Gamma_h} \\ + (\boldsymbol{u}, \nabla q)_{\Omega_h} - \langle \widehat{\boldsymbol{u}} \cdot \boldsymbol{n}, q \rangle_{\Gamma_h} \\ + (\boldsymbol{\sigma}, \boldsymbol{\tau})_{\Omega_h} + (\boldsymbol{u}, \nabla \cdot \boldsymbol{\tau})_{\Omega_h} - \langle \widehat{\boldsymbol{u}}, \boldsymbol{\tau n} \rangle_{\Gamma_h} = (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h} = l(v). \end{aligned} \tag{3.2.1}$$

We begin by identifying the variables involved; we have scalar field variables $u_1, u_2, p, \sigma_{11}, \sigma_{12}, \sigma_{21}, \sigma_{22}$, traces $\widehat{u}_1, \widehat{u}_2$, and fluxes $\widehat{t}_{1n}, \widehat{t}_{2n}$. The test functions are $v_1, v_2, q \in H^1$ and $\boldsymbol{\tau}_1, \boldsymbol{\tau}_2 \in H(\mathrm{div})$. We declare an instance of the `VarFactory` class, which keeps track of the trial and test space variables, and use this to create `VarPtr`s corresponding to each of these variables:

```
VarFactory varFactory;
// traces
VarPtr u1hat = varFactory.traceVar("\\widehat{u}_1");
VarPtr u2hat = varFactory.traceVar("\\widehat{u}_2");
// fluxes
VarPtr t1n = varFactory.fluxVar("\\widehat{t}_{1n}");
VarPtr t2n = varFactory.fluxVar("\\widehat{t}_{2n}");
// fields
VarPtr u1 = varFactory.fieldVar("u_1");
VarPtr u2 = varFactory.fieldVar("u_2");
VarPtr sigma11 = varFactory.fieldVar("\\sigma_11");
```

```
VarPtr sigma12 = varFactory.fieldVar("\\sigma_12");
VarPtr sigma21 = varFactory.fieldVar("\\sigma_21");
VarPtr sigma22 = varFactory.fieldVar("\\sigma_22");
VarPtr p = varFactory.fieldVar("p");
// test functions
VarPtr tau1 = varFactory.testVar("\\tau_1", HDIV);  // tau_1
VarPtr tau2 = varFactory.testVar("\\tau_2", HDIV);  // tau_2
VarPtr v1 = varFactory.testVar("v_1", HGRAD); // v_1
VarPtr v2 = varFactory.testVar("v_2", HGRAD); // v_2
VarPtr q = varFactory.testVar("q", HGRAD); // q
```

The strings in each variable definition are human- and/or TeX-readable identifiers for each variable, which can be used in debugging output, or to generate, say, TeX tabulations of numerical results. The `Var` class stores a unique ID for each variable, the function space for the variable, and allows various operators to be applied to the variable (first order differential operators and operators involving the unit normal along the element boundary).

Next, we use `varFactory` to create a bilinear form; that is, a `BF` object.

```
BFPtr stokesBF = Teuchos::rcp( new BF(varFactory) );
double mu = 1.0;
// v1 terms:
stokesBF->addTerm(sigma11 - p, v1->dx()); // (\sigma_1 - \binom{p}{0}, \nabla v_1)
stokesBF->addTerm(sigma12, v1->dy());
stokesBF->addTerm( t1n, v1);

// v2:
stokesBF->addTerm(sigma21, v2->dx());     // (\sigma_2 - \binom{0}{p}, \nabla v_2)
stokesBF->addTerm(sigma22 - p, v2->dy());
stokesBF->addTerm( t2n, v2);

// q:
stokesBF->addTerm(-u1, q->dx()); // (-u, \nabla q)
stokesBF->addTerm(-u2, q->dy());
stokesBF->addTerm(u1hat->times_normal_x()
                + u2hat->times_normal_y(), q);

// tau1:
```

```
stokesBF->addTerm(u1, tau1->div());
stokesBF->addTerm(sigma11 / mu, tau1->x()); // (1/μ σ1, τ1)
stokesBF->addTerm(sigma12 / mu, tau1->y());
stokesBF->addTerm(-u1hat, tau1->dot_normal());

// tau2:
stokesBF->addTerm(u2, tau2->div());
stokesBF->addTerm(sigma21 / mu, tau2->x()); // (1/μ σ2, τ2)
stokesBF->addTerm(sigma22 / mu, tau2->y());
stokesBF->addTerm(-u2hat, tau2->dot_normal());
```

Thus, in 23 lines of code, we have implemented the entire bilinear form. By way of contrast, an earlier version of Camellia—prior to the introduction of `LinearTerm`—required over 300 lines of code for the same purpose.

We also need to specify the right hand side for our variational form. This is managed by the `RHSEasy` class;[1] right hand sides are linear functionals on the test space, so these can be specified as the sum of `LinearTermPtr` objects. In the case of Stokes cavity flow, the right hand side is zero, but the general form for specifying the right-hand side is as follows:

```
FunctionPtr zero = Function::zero();
Teuchos::RCP<RHSEasy> rhs = Teuchos::rcp( new RHSEasy );
rhs->addTerm( zero * v1 + zero * v2 );
```

[1]The class is so named because an abstract superclass belonging to an earlier design has the name `RHS`; our ultimate plan is to merge the two classes.

3.2.2 Stokes Discretization

Camellia's discretizations are handled by the `Mesh` class;[2] one has simply to define the "H^1 order" in the trial space, given by $k+1$ according to the notation in Chapter 2, and the Δk enrichment for the test space, and Camellia will determine the appropriate orders for each variable.[3]

Our philosophy in DPG is to start with a coarse mesh, and allow adaptivity to refine in regions of largest error. Here, we create a 2×2 initial quadrilateral mesh on a unit square with quadratic field variables and quintic test functions using a convenience constructor provided by `MeshFactory`:

```
int k = 2;
int H1Order = k+1; // L^2 order plus 1
int delta_k = 2;   // test space enrichment
int horizontalCells = 2, verticalCells = 2;
double width = 1.0;
double height = 1.0;
double delta_k = 2;
MeshPtr mesh = MeshFactory::quadMesh(stokesBF, H1Order,
                                     delta_k, width, height,
                                     horizontalCells,
                                     verticalCells);
```

The `mesh` object, on construction, selects appropriate discrete bases according to the functional spaces used in the bilinear form. By default, the basis

[2] It is perhaps worth noting that the `Mesh` class defines the entire discretization: the geometry, element boundaries, *and* the discrete trial and test spaces on each element. This is the reason why the `stokesBF` argument is required here. This is something we hope to change in a redesign, as there are many situations in which it is desirable to solve multiple problems on a single mesh, which the present design makes inconvenient. In lid-driven cavity flow, for example, we solve for streamlines using the solution to the `stokesBF` variational form as data.

[3] There are mechanisms for overriding the default choices if, for instance, one desired order $k+1$ for the velocity field variable and k for the pressure.

functions used are nodal bases (with spectrally distributed nodes) defined by Intrepid. Camellia also provides hierarchical $H(\text{div})$ and H^1 bases derived from the Lobatto (that is, the integrated Legendre) polynomials which may be used for test space discretization on quadrilateral elements. These can be selected by calling the following methods early on[4] in the execution of the driver.

```
BasisFactory::setUseLobattoForQuadHDiv(true);
BasisFactory::setUseLobattoForQuadHGrad(true);
```

See Chapter 9 for some exploration of the effects of the choice of bases on conditioning.

3.2.3 Stokes Boundary Conditions

Camellia defines a `BC` class to allow specification of boundary conditions as well as zero-mean constraints. Let us specify boundary conditions for the lid-driven cavity flow problem described in Section 6.2. Recall that our implementation of the problem approximates the boundary conditions to make them continuous, introducing a "ramp" of width $\epsilon = \frac{1}{64}$. We require:

- along the top boundary, $u_1 = \begin{cases} \frac{x}{\epsilon} & x \leq \epsilon, \\ 1 & \epsilon < x < 1-\epsilon, \\ \frac{1-x}{\epsilon} & 1-\epsilon \leq x, \end{cases}$
- along the left, right, and bottom walls of the cavity, $u_1 = 0$, and

[4]The methods here invoked are static, and the `BasisFactory` class itself is also statically defined, which is why early invocation is important: the invocation should occur before the static `BasisFactory` class is otherwise used. We hope to change this in a future version of Camellia.

- along the entire boundary, $u_2 = 0$.

Furthermore, because the pressure p only enters the formulation through a gradient, it will only be determined up to a constant. To fix its value, we impose a zero-mean constraint, $\int_\Omega p = 0$.

To specify the velocity BCs, we begin by implementing functions defined on the boundary. Camellia provides a `SimpleFunction` class, which is itself a subclass of `Function`: both classes define spatially varying functions; `SimpleFunction` presents a simpler interface that assumes the functions depend *only* on spatial coordinates. We define subclasses of `SimpleFunction` for the BCs on u_1.

```
class U1_0 : public SimpleFunction {
  double _eps;
public:
  U1_0(double eps) {
    _eps = eps;
  }
  double value(double x, double y) {
    double tol = 1e-14;
    if (abs(y-1.0) < tol) { // top boundary
      if ( (abs(x) < _eps) ) { // top left
        return x / _eps;
      } else if ( abs(1.0-x) < _eps) { // top right
        return (1.0-x) / _eps;
      } else { // top middle
        return 1;
      }
    } else { // not top boundary: 0.0
      return 0.0;
    }
  }
};
```

Next, we need a mechanism to specify the portion of the boundary along which each BC is imposed—for example, some BCs might be imposed along an inflow boundary, while others are imposed only along the outflow boundary. Camellia allows the boundary to be restricted by means of a `SpatialFilter` subclass. In the case of lid-driven cavity flow, all our BCs are imposed along the entire boundary, so our `SpatialFilter` subclass simply matches the entire boundary.

```
class UnitSquareBoundary : public SpatialFilter {
public:
  bool matchesPoint(double x, double y) {
    double tol = 1e-14;
    bool xMatch = (abs(x) < tol) || (abs(x-1.0) < tol);
    bool yMatch = (abs(y) < tol) || (abs(y-1.0) < tol);
    return xMatch || yMatch;
  }
};
```

For generality, we have explicitly defined a subclass of `SpatialFilter`, but this particular case could instead have been implemented in a single line of code as follows:

```
  SpatialFilterPtr entireBoundary = SpatialFilter::allSpace();
```

Now, in the main body of the code, we need to create instances of our `Function` and `SpatialFilter` subclasses, as well as a new `BC` object.[5] We then add Dirichlet conditions for $\widehat{u}_1$ and $\widehat{u}_2$, as well as the zero-mean constraint on p. Note that the boundary condition for $\widehat{u}_2$ is defined using `Function::zero()`,

[5]The class we are here, for simplicity of exposition, calling `BC` is in fact implemented as `BCEasy`, because an earlier abstract superclass already has the name `BC`. We expect to rename the classes to match the exposition at some point in the future; the present `BC` class might become `AbstractBC`, for example.

a statically defined `FunctionPtr` subclass; there are several such functions available in `Function`, including constants, select trigonometric functions, mechanisms for "vectorizing" other functions, and more.

```
double eps = 1.0 / 64.0;
BCPtr bc = Teuchos::rcp( new BC );
SpatialFilterPtr entireBoundary =
                        Teuchos::rcp( new UnitSquareBoundary );
FunctionPtr u1_0 = Teuchos::rcp( new U1_0(eps) );
FunctionPtr u2_0 = Function::zero();

bc->addDirichlet(u1hat, entireBoundary, u1_0);
bc->addDirichlet(u2hat, entireBoundary, u2_0);
bc->addZeroMeanConstraint(p);
```

3.2.4 Stokes Test Space Inner Product

In Chapter 2, we outlined a procedure for determining the graph norm from a bilinear form. Camellia uses this very procedure to determine the graph norm automatically, requiring just a single line of code:

```
IPPtr ip = bf->graphNorm();
```

Suppose another norm is desired—for example, in Chapter 6, we will discuss a weighted graph norm motivated by a scaling argument, given by

$$\begin{aligned}\|(\boldsymbol{v}, q, \boldsymbol{\tau})\|_V^2 := &\|\nabla \cdot \boldsymbol{v}\|^2 + \|\nabla \boldsymbol{v} + \boldsymbol{\tau}\|^2 \\ &+ \|h\nabla \cdot \boldsymbol{\tau} - h\nabla q\|^2 + \left\|\frac{\boldsymbol{v}}{h}\right\|^2 + \|q\|^2 + \|\boldsymbol{\tau}\|^2,\end{aligned}$$

where h is the local element diameter. Camellia's `IP` class allows definition of an inner product in terms of linear terms—by calling `addTerm()` with a `LinearTermPtr` argument, one can indicate that the square of the desired

norm should include as summand the square of the specified term. Thus, one can specify the weighted graph norm in Camellia as follows:

```
FunctionPtr h = Function::h(); // element diameter
IPPtr ip = Teuchos::rcp( new IP ); // create inner product

ip->addTerm( v1->dx() + v2->dy() );  // pressure
ip->addTerm( v1->dx() + tau1->x() ); // sigma11 term
ip->addTerm( v1->dy() + tau1->y() ); // sigma12
ip->addTerm( v2->dx() + tau2->x() ); // sigma21
ip->addTerm( v2->dy() + tau2->y() ); // sigma22
ip->addTerm( h * tau1->div() - h * q->dx() ); // u1
ip->addTerm( h * tau2->div() - h * q->dy());  // u2

// L^2 terms:
ip->addTerm( v1 / h );
ip->addTerm( v2 / h );
ip->addTerm( q );
ip->addTerm( tau1 );
ip->addTerm( tau2 );
```

3.2.5 Solving

The effort of determining optimal test functions, assembling the global stiffness matrix, and solving for the DPG degrees of freedom is encapsulated in the `Solution` class. Moreover, the optimal test function determination and stiffness matrix assembly—both of which are embarrassingly parallel operations—are performed in a distributed fashion, using a spatially-local mesh partitioning provided by the `Mesh` class (the `Mesh` class in turn uses Trilinos's Zoltan package for constructing the partitioning). One can construct a `SolutionPtr` object and solve as follows:[6]

[6]One can solve either with or without static condensation; generally, static condensation is recommended. In addition to being faster, the condensed stiffness matrix generally is

```
SolutionPtr solution = Teuchos::rcp( new Solution(mesh, bc,
                                                  rhs, ip) );
solution->condensedSolve();
```

After the solve is complete, there are mechanisms for constructing `FunctionPtr` objects based on the `SolutionPtr`, which can be used for further computations. There are also mechanisms for outputting the solution to a VTK file for visualization.

3.3 Adaptivity

Camellia provides a simple mechanism for h-, p-, and hp-adaptivity using a greedy refinement strategy through the `RefinementStrategy` class. Given a `SolutionPtr` object `solution`, one may define a greedy refinement strategy by specifying

```
double threshold = 0.20;
RefinementStrategy refStrategy( solution, threshold );
```

where the value of 0.20 indicates that elements with error greater than 20% of the maximum element error will be refined in a given refinement step. The default implementation performs h-refinements. One may implement other refinement types by subclassing `RefinementStrategy` and overriding the method

```
virtual void refineCells(vector<int> &cellIDs);
```

better conditioned. However, Lagrange constraints, which are supported by the standard solve (and which are useful for, say, enforcing local conservation), are not yet supported by the static condensation solve.

The superclass then handles the determination of which cells should be refined, and the subclass simply needs to determine what sorts of refinements should be performed on a per-cell basis. Once a refinement strategy is in hand and a solve is completed, a refinement step can be accomplished in a single line of code:

```
refStrategy.refine();
```

3.4 Riesz Representations

As we saw in Chapter 2, Riesz representation functions play a crucial role in DPG; an optimal test function, for example, is precisely the Riesz representation of a bilinear form in which the trial space argument has been fixed, understood as a functional on the test space. Similarly, the error representation function used to determine the energy error and drive adaptivity is the Riesz representation of the residual. Recognizing that the `LinearTerm` class defines functionals on the test and trial spaces, Camellia allows Riesz representations to be defined in terms of `LinearTermPtr` objects, together with an `IPPtr` object representing the inner product relative to which the Riesz representation should be taken, and a `MeshPtr` object corresponding to the mesh on which the representation should be computed. Riesz representations are vector-valued, with components corresponding to each variable in the test (or trial) space. Camellia provides a `Function` subclass, `RepFunction`, which allows one to perform arbitrary computations with the component of the Riesz representation corresponding to a variable in the test (trial) space. The code

below demonstrates construction of a such a function corresponding to the `var` component of a functional `linearTerm`.

```
RieszRepPtr rieszRep = Teuchos::rcp(
                        new RieszRep(mesh, ip, linearTerm) );
FunctionPtr repFxn = Teuchos::rcp(
                        new RepFunction( var, rieszRep) );
```

3.5 Nonlinear Problems

For the Navier-Stokes equations, we will employ a bilinear form based on the linearized Navier-Stokes equations. This will then be solved by means of a Newton-Raphson iteration, using the L^2 norm of field variables in the incremental solution as a stopping criterion. Thus we require mechanisms for:

- representing previous solutions as material data,
- evaluating L^2 norms of the incremental solution, and
- summing the previous solution and the incremental solution.

Camellia provides convenient mechanisms for each of these. The first two come by way of the static method `Function::solution(VarPtr var, SolutionPtr soln)` method, which returns a `FunctionPtr` representing `var` component of solution `soln`. The third is handled by a method provided by the `Solution` class, `addSolution(SolutionPtr soln)`. Each of these is demonstrated in the code snippets below, which assume that the `SolutionPtr` object `prevSoln` represents the background flow (that is, the accumulated solution) and `incrSoln` represents the incremental solution.

```
// ...
// define previous solution function for x-velocity u1:
FunctionPtr u1_prev = Function::solution(u1, prevSoln);
// add convective terms with u1_prev to bilinear form:
navierStokesForm->addTerm( - Re * u1_prev * sigma11, v1);
navierStokesForm->addTerm( - Re * u1_prev * sigma21, v2);
// ...
incrSoln->solve();
// add incremental solution to prevSoln
prevSoln->addSolution(incrSoln);
// check whether we're done:
FunctionPtr u1_incr = Function::solution(u1, incrSoln);
// ...
FunctionPtr fiedsSquared = u1_incr * u1_incr
                         + u2_incr * u2_incr
                         + p_incr * p_incr
                         + sigma11_incr * sigma11_incr
                         + sigma12_incr * sigma12_incr
                         + sigma21_incr * sigma21_incr
                         + sigma22_incr * sigma22_incr;
double l2incr = fieldsSquared->integrate(incrSoln->mesh());
if (l2incr < 1e-8) {
  break;
}
// ...
```

3.6 Curvilinear Meshes

As discussed in Chapter 2, when using curved geometries, the order of finite element solution convergence is generally limited by the order of approximation of the geometry. Thus to achieve higher-order convergence we require higher-order geometry representations. Several requirements present themselves:

1. Provide a mechanism for *specifying* curvilinear geometry. Because we may vary the polynomial order, the best way to approach this is by

allowing exact specification of the geometry, rather than any particular polynomial approximation thereof.

2. Internally, compute a polynomial approximation of the geometry.

3. Internally, account for the curved mesh geometry by changing Jacobians to account for the polynomial geometry approximation, as well as recomputing geometry on mesh refinement.

At present, Camellia supports curvilinear quadrilateral elements. We hope to support triangles in a similar fashion.

3.6.1 Geometry Specification

Most elements in the mesh will not require curved edges; only those that are on the boundary may need curved edges. For this reason, we first define a straight-edge mesh, then allow any edge to be replaced by a parametrically specified curve $\boldsymbol{f}(t)$. The mesh requires that $\boldsymbol{f}(0)$ interpolates the first vertex in the edge and $\boldsymbol{f}(1)$ interpolates the second. Camellia provides several mechanisms for specifying parametric functions—the core class is the `ParametricCurve` class, instances of which can be defined either through subclassing or by specifying the x and y components as `FunctionPtr`s, in which the x variable is taken as the parametric argument. For example, to specify a unit circle centered at the origin, one might write:

```
FunctionPtr cos_2pi_t = Teuchos::rcp( new Cos_ax(2.0*PI) );
FunctionPtr sin_2pi_t = Teuchos::rcp( new Sin_ax(2.0*PI) );
ParametricCurvePtr circle = ParametricCurve::curve(cos_2pi_t,
                                                   sin_2pi_t);
```

`ParametricCurve` allows definition a parametric curve as a segment of another via its `subCurve` method; for example, to define the quarter circle in the first quadrant of the plane, one could write:

```
ParametricCurvePtr arc = ParametricCurve::subCurve(circle,0,
                                                   0.25);
```

`ParametricCurve` also provides built-in definitions of line segments, circles, and circular arcs.

Once the parametric curves are defined, one only has to associate these with edges in the mesh. Suppose that one has a mesh whose cell 0's first edge goes from (1,0) to (0,1), and that the curvilinear mesh desired is this mesh with the that edge replaced by the arc defined above. Executing that replacement is accomplished by the following code:

```
int cellID = 0;
vector<int> vertices = mesh->vertexIndicesForCell(cellID);
pair<int,int> edge = make_pair(vertices[0],vertices[1]);
map< pair<int,int>, ParametricCurvePtr > edgeToCurveMap;
edgeToCurveMap[edge] = arc;
mesh->setEdgeToCurveMap(edgeToCurveMap);
```

3.6.2 Accounting for Curved Geometry in Computations

Transparent to the user writing a driver using Camellia, there are several choices and mechanisms relating to the use of curved geometry that we relate here. Because of the transparency to the user—refinements for curved elements, for example, are invoked in precisely the same way as for straight-edged ones—in this subsection, we omit any sample code.

3.6.2.1 Geometry Approximation

We use the specified curves to compute a two-dimensional *transfinite interpolant* which exactly matches the edges.

We take an isoparametric approach to geometry approximation, in that we use exactly our $\boldsymbol{H}^1$ basis to represent the transformation from the straight-edge mesh to the curvilinear one. This transformation is determined using *projection-based interpolation* of the transfinite interpolant.

In order to avoid limiting the approximation of the test functions, we use the test degree as the order of our geometric approximation.

3.6.2.2 Jacobians

We already compute a Jacobian, an inverse Jacobian, and a Jacobian determinant for the straight-edge mesh. To compute the Jacobian from reference to physical space, we simply need to multiply the existing Jacobian by the Jacobian of our transformation from straight-edge to physical space. The resulting Jacobian (and its inverse and determinant) can then replace the existing in all computations.

3.6.2.3 Refinements

When an element with a curved edge is refined, we need to recreate the curve. Because of the parametric nature of the curve, this only requires remapping the input—and this can be done algebraically, without any need for recursion. (We do also have to compute the new vertex according to the

exact geometry.) We can then use the existing apparatus to recompute the transformation function for the newly created elements.

At present, all newly introduced edges are straight lines with the newly introduced vertices corresponding to the exact geometry. This allows us to minimize the computational cost associated with curved geometry (by keeping the number of elements with curved edges to a minimum); however, we may get better results by instead introducing curved edges corresponding to the transfinite interpolant. This will require defining "patches" of the transfinite interpolant, which is just the two dimensional analog to subdivisions of parametric curves; by similar logic, it should be possible to define these without resorting to a recursive strategy.

As mentioned in Chapter 2 and documented in Chapter 5, the evidence to date suggests that our convergence rates do not suffer due to these straight-edged-interior refinements, and Camellia can take advantage of some savings in computation thanks to this choice—and the amount of savings increases as the mesh is refined: elements with curves will lie only along the curved boundaries. However, a full exploration of the costs and benefits of our approach to refinements—as well as support for more general geometries, including highly curved elements—will require implementation of the more standard approach (refining with curved edges in the interior); we hope to add such a feature to Camellia in the future.

Chapter 4

Analysis of the Velocity-Gradient-Pressure Stokes Formulation

The purpose of this chapter is to present a general theory for the DPG method and apply it to the Stokes problem. This analysis recapitulates results obtained in [36, 12, 38] for particular boundary-value problems and specializes the general theory for Friedrichs systems, presented in [19], for cases in which the traces of the graph spaces are available.

We begin in Section 4.1 by establishing notation, definitions, and some basic properties of our functional setting. Then, in Section 4.2, we discuss, beginning with an abstract first-order linear operator, the strong and ultra-weak formulations of an arbitrary linear PDE, culminating in a proof of the well-posedness of the ultra-weak formulation. Along the way, we highlight some key assumptions that we make on the PDE, and verify those assumptions in the context of the velocity-gradient-pressure (VGP) Stokes formulation. Up to that point, we employ a continuous test space; in Section 4.3 we establish well-posedness in the context of the broken test spaces which are at the heart of DPG—these are what enable tractable determination of the optimal test functions. In Section 4.4 we discuss details of the boundary

conditions and functional setting (including mechanisms for treating the pressure space) for the Stokes problem, expanding further on its satisfaction of our various assumptions. We summarize the results in Section 4.5. Finally, for completeness, in Section 4.6 we establish that the strong operator on which the VGP formulation is based is in fact bounded below, a key condition required by the abstract analysis.

4.1 Notation and Definitions

Let Ω denote a bounded Lipschitz domain in $\mathbb{R}^n$, for $n = 2$ or 3, with boundary $\Gamma = \partial\Omega$. We will employ the standard energy spaces

$$\begin{aligned} H^1(\Omega) &:= \{u \in L^2(\Omega) \,:\, \nabla u \in \boldsymbol{L}^2(\Omega)\}, \\ H(\mathrm{div},\Omega) &:= \{\boldsymbol{\sigma} \in \boldsymbol{L}^2(\Omega) \,:\, \nabla\cdot\boldsymbol{\sigma} \in L^2(\Omega)\}, \end{aligned}$$

with corresponding trace spaces on Γ

$$\begin{aligned} H^{1/2}(\Gamma) &:= \{\hat{u} = u|_\Gamma,\ u \in H^1(\Omega)\}, \\ H^{-1/2}(\Gamma) &:= \{\hat{\sigma}_n = (\boldsymbol{\sigma}\cdot\boldsymbol{n})|_\Gamma,\ \boldsymbol{\sigma} \in H(\mathrm{div},\Omega)\}, \end{aligned}$$

where $\boldsymbol{n}$ denotes the outward normal unit vector to the boundary Γ. The assumption that the domain is Lipschitz is essential; domains with cracks require a special and non-classical treatment. We define a trace operator

$$\mathrm{tr} \,:\, H^1(\Omega) \ni u \to \mathrm{tr}\ u = \hat{u} = u|_\Gamma \in H^{1/2}(\Gamma).$$

The space $H^{-1/2}(\Gamma)$ is the topological dual of $H^{1/2}(\Gamma)$ and we similarly define a trace operator

$$\mathrm{tr} \,:\, H(\mathrm{div},\Omega) \ni \boldsymbol{\sigma} \to \mathrm{tr}\ \boldsymbol{\sigma} = \hat{\sigma}_n = (\boldsymbol{\sigma}\cdot\boldsymbol{n})|_\Gamma \in H^{-1/2}(\Gamma).$$

Note that, unless otherwise stated, in this chapter we use the same trace notation "tr" for functions in both $H^1(\Omega)$ and $H(\mathrm{div}, \Omega)$.

We shall also use group variables whose components belong to $H^1(\Omega)$, $H(\mathrm{div}, \Omega)$, $H^{1/2}(\Gamma)$ or $H^{-1/2}(\Gamma)$. We will employ boldface notation to distinguish the Cartesian product spaces from their scalar counterparts:

$$\begin{aligned} \boldsymbol{H}^1(\Omega) &= H^1(\Omega) \times \ldots \times H^1(\Omega), \\ \boldsymbol{H}^{1/2}(\Gamma) &= H^{1/2}(\Gamma) \times \ldots \times H^{1/2}(\Gamma), \\ \boldsymbol{H}^{-1/2}(\Gamma) &= H^{-1/2}(\Gamma) \times \ldots \times H^{-1/2}(\Gamma), \end{aligned}$$

etc. In the case of tensors, the definitions will be applied row-wise:

$$\boldsymbol{\sigma} = (\sigma_{ij}) \in \boldsymbol{H}(\mathrm{div}, \Omega) \quad \Longleftrightarrow \quad (\sigma_{i1}, \ldots, \sigma_{in}) \in H(\mathrm{div}, \Omega),\ i = 1, \ldots, n.$$

Broken energy spaces. Let Ω be partitioned into finite elements K such that

$$\overline{\Omega} = \bigcup_K \bar{K}, \quad K \text{ open},$$

with a corresponding *skeleton* Γ_h and *interior skeleton* Γ_h^0,

$$\Gamma_h := \bigcup_K \partial K \qquad \Gamma_h^0 := \Gamma_h \setminus \Gamma.$$

The elements may be general polygons in two dimensions, or polyhedra[1] in three (with triangular and quadrilateral faces). Meshes may be *irregular*, i.e. may contain hanging nodes (see e.g. [34, pp. 211]). At this point we make

[1] Possibly curvilinear polyhedra.

no shape regularity assumptions. We define *broken* energy spaces, which are simply standard energy spaces defined element-wise:

$$\begin{aligned} H^1(\Omega_h) &:= \textstyle\prod_K H^1(K), \\ H(\mathrm{div}, \Omega_h) &:= \textstyle\prod_K H(\mathrm{div}, K). \end{aligned}$$

In the broken energy spaces, integration by parts is performed element-wise. For $\boldsymbol{\sigma} \in H(\mathrm{div}, \Omega_h)$ and $v \in H^1(\Omega)$, we have

$$\begin{aligned} (\mathrm{div}_h \boldsymbol{\sigma}, v)_{\Omega_h} :=& \sum_K (\mathrm{div}\boldsymbol{\sigma}, v)_K \\ =& \sum_K (-(\boldsymbol{\sigma}, \nabla v)_K + \langle \hat{\sigma}_n, \hat{v} \rangle_{\partial K}) \\ =& -(\boldsymbol{\sigma}, \nabla v) + \underbrace{\sum_K \langle \hat{\sigma}_n, \hat{v} \rangle_{\partial K}}_{=: \langle \hat{\sigma}_n, \hat{v} \rangle_{\Gamma_h}}. \end{aligned}$$

Here $(\cdot, \cdot)$ and $(\cdot, \cdot)_K$ denote the L^2 inner product over the domain Ω and the element K, respectively, and $\langle \cdot, \cdot \rangle_{\partial K}$ stands for the duality pairing between $H^{-1/2}(\partial K)$ and $H^{1/2}(\partial K)$.

This leads us naturally to the concept of the trace space on the skeleton Γ_h:

$$H^{1/2}(\Gamma_h) := \left\{ \hat{v} = \{\hat{v}_K\} \in \prod_K H^{1/2}(\partial K) \ : \ \exists v \in H^1(\Omega) : v|_{\partial K} = \hat{v}_K \right\}.$$

There are a couple of important subtleties here. First, by $v|_{\partial K}$ we mean the trace (for element K) of the restriction of v to K. Second, $H^{1/2}(\Gamma_h)$ is a *closed* subspace of $\prod_K H^{1/2}(\partial K)$, as we show below. We take the trace spaces to be endowed with *minimum-energy extension norms*, given by

$$\|\hat{u}\|_{H^{1/2}(\partial K)} := \inf_{\substack{u \in H^1(K) \\ u|_{\partial K} = \hat{u}}} \|u\|_{H^1(K)} \quad \text{and} \quad \|\widehat{\sigma}_n\|_{H^{-1/2}(\partial K)} := \inf_{\substack{\boldsymbol{\sigma} \in H(\mathrm{div}, K) \\ (\boldsymbol{\sigma} \cdot \boldsymbol{n})|_{\partial K} = \widehat{\sigma}_n}} \|u\|_{H^1(K)}.$$

To see that $H^{1/2}(\Gamma_h)$ is a closed subspace of the product space, let $\hat{u}^n = \{\hat{u}_K^n\} \in H^{1/2}(\Gamma_h)$ be a sequence of functions such that

$$\hat{u}^n \xrightarrow[\prod_K H^{1/2}(\partial K)]{} \hat{u} = \{\hat{u}_K\},$$

where $\hat{u}$ is a member of the product space. We seek to show that there exists $u = \{u_K\} \in H^1(\Omega)$ such that $u|_{\partial K} = \hat{u}_K$ for each K, where again by $u|_{\partial K}$ we denote the element boundary trace of the restriction of u to element K. Now, for each K, $\hat{u}_K^n = u^n|_{\partial K}$ for some $u^n \in H^1(\Omega)$. By the definition of norms, $u^n|_K \overset{n\to\infty}{\longrightarrow} u_K$ in $H^1(K)$, for each element K. Now, the delicate question is whether we can claim that the union $u = \{u_K\}$ is in $H^1(\Omega)$. We can show this by the definition of distributional derivatives. Consider a test function $\phi \in \mathcal{D}(\Omega)$. For each n, the distributional derivative $\frac{\partial u^n}{\partial x_i}$ is defined by

$$\int_\Omega u^n \frac{\partial \phi}{\partial x_i} = -\int_\Omega \frac{\partial u^n}{\partial x_i}\phi$$

or, equivalently,

$$\sum_K \int_K u^n \frac{\partial \phi}{\partial x_i} = -\sum_K \int_K \frac{\partial u^n}{\partial x_i}\phi.$$

Passing to the limit $n \to \infty$, we have

$$\int_\Omega u \frac{\partial \phi}{\partial x_i} = -\sum_K \int_K \frac{\partial u_K}{\partial x_i}\phi,$$

which shows that the union of element-wise derivatives $\left\{\frac{\partial u_K}{\partial x_i}\right\}$ *is* the distributional derivative of u. Now, since $u_K \in H^1(K)$, $\frac{\partial u_K}{\partial x_i} \in L^2(K)$, so that the union $\left\{\frac{\partial u_K}{\partial x_i}\right\} \in L^2(\Omega)$. Consequently, $u \in H^1(\Omega)$.

A similar construction holds for globally conforming $\boldsymbol{\sigma} \in H(\mathrm{div}, \Omega)$ but broken $v \in H^1(\Omega_h)$:

$$\begin{aligned}(\boldsymbol{\sigma}, \nabla_h v)_{\Omega_h} &:= \sum_K (\boldsymbol{\sigma}, \nabla v)_K \\ &= \sum_K \left(-(\nabla \cdot \boldsymbol{\sigma}, v)_K + \langle \hat{\sigma}_n, \hat{v} \rangle_{\partial K}\right) \\ &= -(\nabla \cdot \boldsymbol{\sigma}, v) + \underbrace{\sum_K \langle \hat{\sigma}_n, \hat{v} \rangle_{\partial K}}_{=: \langle \hat{\sigma}_n, \hat{v} \rangle_{\Gamma_h}}.\end{aligned}$$

In this case, we are led to the definition of the trace space $H^{-1/2}(\Gamma_h)$:

$$\begin{aligned}H^{-1/2}(\Gamma_h) := \{\hat{\sigma}_n = \{\hat{\sigma}_{Kn}\} \in \prod_K H^{-1/2}(\partial K) \\ : \exists \boldsymbol{\sigma} \in H(\mathrm{div}, \Omega) : \hat{\sigma}_{Kn} = (\boldsymbol{\sigma} \cdot \boldsymbol{n})|_{\partial K}\}.\end{aligned}$$

We equip both trace spaces over the mesh skeleton with minimum energy extension norms

$$\|\hat{v}\|_{H^{1/2}(\Gamma_h)} := \inf_{\substack{u \in H^1(\Omega) \\ u|_{\Gamma_h} = u}} \|u\|_{H^1(\Omega)} \quad \text{and}$$

$$\|\hat{\sigma}_n\|_{H^{-1/2}(\Gamma_h)} := \inf_{\substack{\boldsymbol{\sigma} \in \boldsymbol{H}(\mathrm{div}, \Omega) \\ (\boldsymbol{\sigma} \cdot \boldsymbol{n})|_{\Gamma_h} = \hat{\sigma}_n}} \|\boldsymbol{\sigma}\|_{\boldsymbol{H}(\mathrm{div}, \Omega)}.$$

We will also need the space of traces on the internal skeleton

$$\tilde{H}^{1/2}(\Gamma_h) := \left\{ \hat{v} = \{\hat{v}_K\} \in \prod_K H^{1/2}(\partial K) \ : \ \exists v \in H_0^1(\Omega) : v|_{\partial K} = \hat{v}_K \right\},$$

which we likewise equip with the minimum energy extension norm.

We have thus defined the term $\langle \hat{\sigma}_n, \hat{v} \rangle_{\Gamma_h}$ when one of the variables is a trace over the whole skeleton and the other is the trace of a function from the broken energy space. For sufficiently regular functions, $\langle \hat{\sigma}_n, \hat{v} \rangle_{\Gamma_h}$ represents

either the $L^2(\Gamma_h)$-product of trace of a conforming $\boldsymbol{\sigma}$ and inter-element jumps of v, or the product of jumps in σ_n and the trace of a globally conforming v. This can be seen by switching from the summation over elements to the summation over element faces (edges in 2D).

4.2 Strong and Ultra-Weak Formulations

We now turn to an abstract setting amenable to analysis. It is worth emphasizing that in this section we work with functions defined on the whole domain Ω and its boundary Γ. Only in Section 4.3 do we once again consider a division of Ω into elements K and broken spaces defined element-wise.

Integration by parts. Let u now represent an abstract group variable consisting of functions defined on the domain Ω, and A be a linear differential operator corresponding to a system of first order PDEs. We start with an abstract integration by parts formula

$$(Au, v) = (u, A^*v) + c(\mathrm{tr}_A u, \mathrm{tr}_{A^*} v), \tag{4.2.1}$$

where $(\cdot,\cdot)$ denotes the $L^2(\Omega)$-inner product, A^* is the formal adjoint operator, c is a bilinear boundary term arising from integration by parts, and for the moment $\langle\cdot,\cdot\rangle$ represents the $L^2(\Gamma)$-inner product on boundary $\Gamma = \partial\Omega$. Obviously, the formula holds under appropriate regularity assumptions, e.g., $u, v \in C^1(\overline{\Omega})$, if all derivatives are understood in the classical sense.

If we assume $u, v \in L^2(\Omega)$ and interpret the derivatives in a distribu-

tional sense, we arrive naturally at the graph energy spaces

$$\begin{aligned} H_A(\Omega) &:= \{u \in L^2(\Omega) \,:\, Au \in L^2(\Omega)\} \quad \text{and} \\ H_{A^*}(\Omega) &:= \{u \in L^2(\Omega) \,:\, A^*u \in L^2(\Omega)\}. \end{aligned}$$

Assumption 1: We take operators A and A^* to be surjections; i.e., given $f \in L^2(\Omega)$, we can always find $u \in H_A(\Omega)$ and $v \in H_{A^*}(\Omega)$ such that $Au = f$ and $A^*v = f$. Roughly speaking, this corresponds to an assumption that neither A nor A^* are, in a sense, degenerate.[2]

With $u \in H_A(\Omega)$ and $v \in H_{A^*}(\Omega)$, the domain integrals (Au, v) and (u, A^*v) are well-defined. We assume that the graph spaces admit trace operators and trace spaces

$$\begin{aligned} \mathrm{tr}_A &: H_A(\Omega) \twoheadrightarrow \widehat{H}_A(\Gamma) \quad \text{and} \\ \mathrm{tr}_{A^*} &: H_{A^*}(\Omega) \twoheadrightarrow \widehat{H}_{A^*}(\Gamma). \end{aligned}$$

The double arrowheads indicate that the trace operators are surjective. We equip the trace spaces with the minimum energy extension norms

$$\|\hat{u}\|_{\widehat{H}_A(\Gamma)} = \inf_{\substack{u \in H_A(\Omega) \\ \mathrm{tr}_A u = \hat{u}}} \|u\|_{H_A(\Omega)} \quad \text{and} \quad \|\hat{v}\|_{\widehat{H}_{A^*}(\Gamma)} = \inf_{\substack{v \in H_{A^*}(\Omega) \\ \mathrm{tr}_{A^*} v = \hat{v}}} \|v\|_{H_{A^*}(\Omega)}.$$

We now generalize the classical integration by parts formula (4.2.1) to a more general, distributional case:

$$(Au, v) = (u, A^*v) + c(\mathrm{tr}_A u, \mathrm{tr}_{A^*} v),$$

with $u \in H_A(\Omega), v \in H_{A^*}(\Omega)$, and

$$c(\hat{u}, \hat{v}), \quad \hat{u} \in \widehat{H}_A(\Gamma), \hat{v} \in \widehat{H}_{A^*}(\Gamma)$$

[2] Ivo Babuška, private communication.

being a duality pairing, i.e. a *definite* continuous bilinear (sesquilinear) form.[3] Recall that form $c(\hat{u}, \hat{v})$ is definite if

$$(c(\hat{u}, \hat{v}) = 0 \quad \forall \hat{v}) \implies \hat{u} = 0 \quad \text{and}$$

$$(c(\hat{u}, \hat{v}) = 0 \quad \forall \hat{u}) \implies \hat{v} = 0.$$

Equivalently, the corresponding boundary operator

$$C \; : \; \widehat{H}_A(\Gamma) \to (\widehat{H}_{A^*}(\Gamma))', \quad \langle C\hat{u}, \hat{v} \rangle = c(\hat{u}, \hat{v})$$

and its adjoint C' are injective and therefore both C and C' are isomorphisms.[4] From here forward, $\langle \cdot, \cdot \rangle$ denotes the usual duality pairing between a space and its dual.

Integration by parts for the Stokes problem. We verify (and illustrate) our general assumptions for the Stokes problem. Recalling the first order system introduced in Chapter 2 and introducing additional loads g and $\boldsymbol{h}$ on the right-hand side for the purposes of our analysis

$$\begin{aligned} -\nabla p + \nabla \cdot \boldsymbol{\sigma} &= \boldsymbol{f} \\ \nabla \cdot \boldsymbol{u} &= g \\ \boldsymbol{\sigma} - \nabla \boldsymbol{u} &= \boldsymbol{h}. \end{aligned}$$

[3] Here we extend the definition of the usual duality pairing between a space and its dual.
[4] $\mathcal{R}(C) = \mathcal{N}(C')^{\perp}$.

Multiplying by test functions $\boldsymbol{v}, q$, and $\boldsymbol{\tau}$, integrating by parts, and adding the equations, we obtain

$$\begin{aligned}(-\nabla\cdot(\boldsymbol{\sigma}-p\boldsymbol{I}),\boldsymbol{v})+(\nabla\cdot\boldsymbol{u},q)+(\boldsymbol{\sigma}-\nabla\boldsymbol{u},\boldsymbol{\tau})&\\ =\quad(\boldsymbol{\sigma}-p\boldsymbol{I},\nabla\boldsymbol{v})+\langle(-\boldsymbol{\sigma}+p\boldsymbol{I})\boldsymbol{n},\boldsymbol{v}\rangle&\\ +(\boldsymbol{u},-\nabla q)+\langle\boldsymbol{u}\cdot\boldsymbol{n},q\rangle&\\ +(\boldsymbol{\sigma},\boldsymbol{\tau})+(\boldsymbol{u},\nabla\cdot\boldsymbol{\tau})+\langle\boldsymbol{u},-\boldsymbol{\tau}\boldsymbol{n}\rangle&\\ =\quad(\boldsymbol{u},\nabla\cdot(\boldsymbol{\tau}-q\boldsymbol{I}))+(p,-\nabla\cdot\boldsymbol{v})+(\boldsymbol{\sigma},\boldsymbol{\tau}+\nabla\boldsymbol{v})&\\ +\langle(-\boldsymbol{\sigma}+p\boldsymbol{I})\boldsymbol{n},\boldsymbol{v}\rangle+\langle\boldsymbol{u},(-\boldsymbol{\tau}+q\boldsymbol{I})\boldsymbol{n}\rangle.&\end{aligned}$$

To connect this to the abstract discussion above, we define group variables

$$u=(\boldsymbol{u},p,\boldsymbol{\sigma})\quad\text{and}\quad v=(\boldsymbol{v},q,\boldsymbol{\tau}).$$

The operators A and A^* are then defined by

$$Au=(-\nabla\cdot(\boldsymbol{\sigma}-p\boldsymbol{I}),\nabla\cdot\boldsymbol{u},\boldsymbol{\sigma}-\nabla\boldsymbol{u}),$$

$$A^*v=(\nabla\cdot(\boldsymbol{\tau}-q\boldsymbol{I}),-\nabla\cdot\boldsymbol{v},\boldsymbol{\tau}+\nabla\boldsymbol{v}).$$

The operator A is not formally self-adjoint but the corresponding energy graph spaces are identical; $H_A(\Omega)=H_{A^*}(\Omega)$, where

$$H_A(\Omega)=\{(\boldsymbol{u},p,\boldsymbol{\sigma})\ :\ \boldsymbol{\sigma}-p\boldsymbol{I}\in\boldsymbol{H}(\mathrm{div},\Omega),\boldsymbol{u}\in\boldsymbol{H}^1(\Omega)\}.$$

The trace tr_A is given by

$$\mathrm{tr}_A:H_A(\Omega)\ni(\boldsymbol{u},p,\boldsymbol{\sigma})\to((-\boldsymbol{\sigma}+p\boldsymbol{I})\boldsymbol{n},\boldsymbol{u})\in\boldsymbol{H}^{-1/2}(\Gamma)\times\boldsymbol{H}^{1/2}(\Gamma);$$

the trace tr_A^* is similar. The boundary term, being the sum of standard duality pairings for respective components of the traces, is definite.

The Closed Range Theorem. Before we proceed to considering the boundary operator C in the abstract setting, we recall the Banach Closed Range Theorem [16, p. 39], which we will need in what follows.

THEOREM 1

(Closed Range Theorem)

Let $T : X \to Y$ be a linear, continuous operator from a Hilbert space X into a Hilbert space Y. Let $\check{T}$ be the corresponding operator defined on the quotient space $X/\mathcal{N}(T)$ or, equivalently, the restriction of T to the X-orthogonal complement of the null space of operator T. Let $\check{T}^*$ denote the analogous operator for the adjoint T^*. The following conditions are then equivalent to each other.

- *T has a closed range.*
- *T^* has a closed range.*
- $\mathcal{N}(T)^\perp = \mathcal{R}(T^*)$.
- $\mathcal{N}(T^*)^\perp = \mathcal{R}(T)$.
- *$\check{T}$ is bounded below—that is, $\|Tu\| \geq \gamma\|u\| \quad \forall u \in \mathcal{N}(T)^\perp$.*
- *$\check{T}^*$ is bounded below—that is, $\|T^*v\| \geq \gamma\|v\| \quad \forall v \in \mathcal{N}(T^*)^\perp$.*

Note that the (maximal) constant γ is the same for T and T^*.

Strong formulation with homogeneous boundary conditions. Returning to the abstract setting, let us consider the boundary operator C. We assume C can be split into two operators C_1 and C_2 such that

$$\begin{aligned}\langle Cu, v\rangle &= \langle C_1 u, v\rangle + \langle C_2 u, v\rangle \\ &= \langle C_1 u, v\rangle + \langle u, C_2' v\rangle,\end{aligned}$$

where we also take C_1 and C_2 to be "reasonable" in the sense that both have closed range.[5]

We are interested in solving a non-homogeneous boundary-value problem

$$\left\{\begin{array}{rll} Au &= f & \text{in } \Omega, \\ C_1 \mathrm{tr}_A\, u &= f_D & \text{on } \Gamma, \end{array}\right. \tag{4.2.2}$$

with $f \in L^2(\Omega)$ and $f_D \in \mathcal{R}(C_1)$.

We begin with the homogeneous BC case:

$$\left\{\begin{array}{rll} Au &= f & \text{in } \Omega, \\ C_1 \mathrm{tr}_A\, u &= 0 & \text{on } \Gamma. \end{array}\right. \tag{4.2.3}$$

Introducing the spaces

$$\begin{aligned} U &:= \{u \in H_A(\Omega) : \quad C_1\, \mathrm{tr}_A u = 0\} \quad \text{and} \\ V &:= \{v \in H_{A^*}(\Omega) : \quad C_2'\, \mathrm{tr}_{A^*} v = 0\}, \end{aligned}$$

we see that, if we restrict operators A and A^* to U and V, the boundary term vanishes. However, for A and A^* to be L^2-adjoint,[6] we have to make an

[5] This is analogous to boundary operator splitting due to Friedrichs [48, 43].

[6] See the definition of an adjoint defined on a proper subspace in [69, p. 509].

additional technical assumption:

Assumption 2:

$$(\langle u, C_2' v\rangle = 0 \quad \forall u \ : C_1 u = 0) \implies C_2' v = 0. \tag{4.2.4}$$

That is, the domain of the adjoint operator has to be maximal in the sense that it includes *all* v for which the boundary term vanishes. The following lemma allows us to use this condition to establish a decomposition of the trace space.

Lemma 1

Assume C has been split into C_1 and C_2 that satisfy condition (4.2.4). Each of the following conditions is then equivalent to (4.2.4).

- $\mathcal{N}(C_1)^\perp \cap \mathcal{R}(C_2') = \{0\}$.
- $\mathcal{N}(C_1)^\perp \cap \mathcal{N}(C_2)^\perp = \{0\}$.
- $X = \mathcal{N}(C_2) + \mathcal{N}(C_1)$.

∎

The first condition is simply a restatement of Assumption 2. The proof of the lemma is elementary and involves an application of the Closed Range Theorem (Theorem 1 above). Moreover, when $C : X \to Y$ is an isomorphism, the algebraic sum in the last condition can be upgraded to a direct sum:

$$X = \mathcal{N}(C_2) \oplus \mathcal{N}(C_1). \tag{4.2.5}$$

Indeed, $\mathcal{N}(C_1) \cap \mathcal{N}(C_2) \subset \mathcal{N}(C) = \{0\}$.

Condition (4.2.4) thus decomposes the trace space $\widehat{H}_A(\Gamma)$ into the direct sum of the nullspaces of operators C_1 and C_2;

$$\widehat{H}_A(\Gamma) = \widehat{H}_A^1(\Gamma) \oplus \widehat{H}_A^2(\Gamma), \tag{4.2.6}$$

where $\widehat{H}_A^1(\Gamma) = \mathcal{N}(C_2)$ and $\widehat{H}_A^2(\Gamma) = \mathcal{N}(C_1)$. In other words, for each $\hat{u} \in \widehat{H}_A(\Gamma)$, there exist unique $\hat{u}_1 \in \widehat{H}_A^1(\Gamma)$ and $\hat{u}_2 \in \widehat{H}_A^2(\Gamma)$ such that

$$\hat{u} = \hat{u}_1 + \hat{u}_2,$$

which is analogous to the condition introduced by Friedrichs [48] on his boundary operator M, subsequently generalized in [43], and first used in the DPG context in [19].

By making one more assumption, we can establish the well-posedness of the homogeneous problem.

Assumption 3: The operator $A|_U$, restricted to the L^2-orthogonal complement $\mathcal{N}(A)^\perp$, is bounded below.

This assumption is exactly what we need to apply Theorem 1 to the homogeneous problem; the problem (4.2.3) and its adjoint counterpart are thus well-posed. More precisely, for each data function f which is L^2-orthogonal to the null space of the adjoint operator, a solution exists, is unique in the orthogonal complement of the null space of the operator (equivalently, in the quotient space), and depends continuously on f. The inverse of the maximal

constant γ is precisely the norm of the inverse operator from $\mathcal{R}(A)$ into $\mathcal{N}(A)^\perp$ (which is equal to the norm of the solution operator from $\mathcal{R}(A^*)$ into $\mathcal{N}(A^*)^\perp$).

Strong formulation with homogeneous boundary conditions for the Stokes problem. Let us make the abstract boundary operators concrete in the context of the Stokes problem, and check our assumptions. We have

$$C_1 u = C_1(\boldsymbol{u}, p, \boldsymbol{\sigma}) = \begin{pmatrix} \mathbf{0} \\ \text{tr } \boldsymbol{u} \end{pmatrix} \quad \text{and} \quad C_2' v = C_2'(\boldsymbol{v}, q, \boldsymbol{\tau}) = \begin{pmatrix} \text{tr } \boldsymbol{v} \\ \mathbf{0} \end{pmatrix}.$$

Condition (4.2.4) is easily satisfied. We have $U = V$, where

$$U = \{(\boldsymbol{u}, p, \boldsymbol{\sigma}) \in (\boldsymbol{L}^2(\Omega) \times L^2(\Omega) \times \boldsymbol{L}^2(\Omega)) : \boldsymbol{\sigma} - p\boldsymbol{I} \in \boldsymbol{H}(\mathbf{div}, \Omega), \boldsymbol{u} \in \boldsymbol{H}_0^1(\Omega)\}.$$

A and A^* have non-trivial null spaces consisting of constant pressures. To ensure uniqueness, we have to restrict ourselves to pressures p and q with zero average; that is,

$$p, q \in L_0^2 := \left\{ q \in L^2(\Omega) \ : \ \int_\Omega q = 0 \right\}.$$

The proof that A and A^* are bounded below invokes the Babuška-Brezzi inf-sup condition; for the reader's convenience we reproduce the classical reasoning in Section 4.6.

Strong formulation with non-homogeneous boundary conditions. We are now ready to consider the case of non-homogeneous boundary conditions. We have

$$(Au, v) - \langle C_1 u, v \rangle = (u, A^* v) + \langle u, C_2' v \rangle \quad u \in H_A(\Omega), v \in H_{A^*}(\Omega).$$

We are interested in the operators

$$\begin{aligned} H_A(\Omega) \ni u \to \quad & (Au, C_1 u) \in \quad L^2(\Omega) \times (\widehat{H}_{A^*}(\Gamma))' \quad \text{and} \\ H_{A^*}(\Omega) \ni v \to \quad & (A^* v, C_2' v) \in \quad L^2(\Omega) \times (\widehat{H}_A(\Gamma))'. \end{aligned}$$

We have the following classical result.

THEOREM 2

Assume that data $f \in L^2(\Omega)$ and $f_D \in \mathcal{R}(C_1)$ satisfy the compatibility condition

$$(f, v) - \langle f_D, v \rangle = 0 \quad \forall v : A^* v = 0, C_2' v = 0.$$

Problem (4.2.2) has a unique solution u in $\mathcal{N}(A)^\perp$ that depends continuously upon the data; i.e. there exists a constant $\tilde{\gamma} > 0$, independent of the data, such that

$$\tilde{\gamma} \|u\|_{H_A(\Omega)} \leq \left(\|f\|^2 + \|f_D\|^2 \right)^{1/2}.$$

The analogous result holds for the adjoint operator (A^*, C_2'). ∎

Proof. Let $\bar{C}_1$ denote the restriction of C_1 to $\widehat{H}_A^1(\Gamma)$. Since $\bar{C}_1$ is then injective and has closed range, it admits a continuous inverse:

$$\|\hat{u}_1\|_{\widehat{H}_A(\Gamma)} = \left\| \bar{C}_1^{-1} f_D \right\|_{\widehat{H}_A(\Gamma)} \leq \frac{1}{\delta} \|f_D\|_{(\widehat{H}_{A^*}(\Gamma))'}.$$

Let $\hat{u} = (\hat{u}_1, 0)$ and let $\tilde{\hat{u}}$ be the minimum-energy extension of $\hat{u}$ in $H_A(\Omega)$. We seek a solution u of the form

$$u = u_0 + \tilde{\hat{u}},$$

where $u_0 \in \mathcal{N}(A|_U)^\perp$ solves the homogeneous BVP (4.2.3) with the modified right-hand side $f - A\tilde{\hat{u}}$. The load $f - A\tilde{\hat{u}}$ satisfies the compatibility condition for the homogeneous case. Indeed,

$$\begin{aligned}(f - A\tilde{\hat{u}}, v) &= (f, v) - (A\tilde{\hat{u}}, v) \\ &= (f, v) - \left((\tilde{\hat{u}}, A^* v) - \langle f_D, v\rangle - \langle \tilde{\hat{u}}, C_2' v\rangle\right) \\ &= (f, v) - \langle f_D, v\rangle = 0,\end{aligned}$$

for each v such that $A^* v = 0$ and $C_2' v = 0$.

We now have

$$\begin{aligned}\|u\|_{H_A}^2 &= \|u_0 + \tilde{\hat{u}}\|_{H_A}^2 \\ &\leq 2\left(\|u_0\|_{H_A}^2 + \|\tilde{\hat{u}}\|_{H_A}^2\right).\end{aligned}$$

Now, observe that

$$\|\tilde{\hat{u}}\|_{H_A(\Omega)} = \|\hat{u}\|_{\widehat{H}_A(\Gamma)} \leq \frac{1}{\delta}\|f_D\|_{(\widehat{H}_{A^*}(\Gamma))'}$$

and that

$$\begin{aligned}\|u_0\|_{H_A}^2 &= \|u_0\|^2 + \|Au_0\|^2 \\ &\leq \left(\frac{1}{\gamma^2} + 1\right)\|Au_0\|^2 \\ &\leq 2\left(\frac{1}{\gamma^2} + 1\right)\left[\|f\|^2 + \frac{1}{\delta^2}\|f_D\|^2\right].\end{aligned}$$

Combining these results ends the proof with

$$\frac{1}{\tilde{\gamma}} \leq 2\max\left\{\sqrt{\frac{1}{\gamma^2} + 1}, \frac{1}{\delta}\sqrt{\frac{1}{\gamma^2} + \frac{3}{2}}\right\}.$$

□

Strong formulation with non-homogeneous boundary conditions for the Stokes problem. Once again turning from the abstract setting to its concrete application to Stokes, we note that the ranges of operators C_1 and C_2' coincide exactly with $\boldsymbol{H}^{1/2}(\Gamma)$. Recall that in our Stokes formulation above we defined a load on the divergence equation $g = \nabla \cdot \boldsymbol{u}$. The strong formulation for the non-homogeneous Stokes problem is well-posed, provided the data g and $\boldsymbol{u}_D$ satisfy the compatibility condition

$$\int_\Omega g = \int_\Gamma \boldsymbol{u}_D \cdot \boldsymbol{n}.$$

The analogous conclusion holds for the adjoint operator.

Ultra-weak (variational) formulation. We are now ready to formulate the *ultra-weak variational formulation* for problem (4.2.2). The steps are as follows.

1. Integrate by parts:

$$(u, A^* v) + \langle C \mathrm{tr}_A u, v \rangle = (f, v).$$

2. Split the boundary operator C into C_1 and C_2 according to the decomposition in (4.2.6):

$$(u, A^* v) + \langle C_1 (\mathrm{tr}_A u)_1, \mathrm{tr}_{A^*} v \rangle + \langle C_2 (\mathrm{tr}_A u)_2, \mathrm{tr}_{A^*} v \rangle = (f, v).$$

3. Apply the boundary condition by moving the known term $C_1(\mathrm{tr}_A u)_1 = f_D$ to the right-hand side:

$$(u, A^* v) + \langle (\mathrm{tr}_A u)_2, C_2' v \rangle = (f, v) - \langle f_D, \mathrm{tr}_{A^*} v \rangle.$$

4. Introduce $\hat{u}_2 = (\mathrm{tr}_A u)_2$ as an independent unknown. The problem then becomes

$$\begin{cases} \text{Find } u \in L^2(\Omega), \hat{u}_2 \in \widehat{H}^2_A(\Gamma) \text{ such that} \\ (u, A^* v) + \langle \hat{u}_2, C_2' v \rangle = (f, v) - \langle f_D, v \rangle \quad \forall v \in H_{A^*}(\Omega). \end{cases} \tag{4.2.7}$$

The bilinear form

$$b((u, \hat{u}_2), v) := (u, A^* v) + \langle \hat{u}_2, C_2' v \rangle = (u, A^* v) + c(\hat{u}_2, v) \tag{4.2.8}$$

generates two associated operators B and B', where

$$b((u, \hat{u}_2), v) = \langle B(u, \hat{u}_2), v \rangle = \langle (u, \hat{u}_2), B' v \rangle.$$

Operator B' corresponds to the strong setting for the adjoint A^* with non-homogeneous BCs;

$$B' v = (A^* v, C_2' v) \in L^2(\Omega) \times (\widehat{H}_A(\Gamma))'.$$

In order to determine the null space of operator B, assume that

$$b((u, \hat{u}_2), v) = 0 \quad \forall v \in H_{A^*}(\Omega).$$

Testing first with $v \in \mathcal{D}(\Omega)$, we see that $Au = 0$. Integrating the first term by parts, and testing with arbitrary v, we learn that $\hat{u}_2 = u$ on Γ and $C_1 u = 0$.

THEOREM 3

Problem (4.2.7) is well-posed. In particular, for each f and f_D which satisfy the compatibility condition

$$(f, v) - \langle f_D, v \rangle = 0 \quad \forall v \in \mathcal{N}(A^*|_V), \tag{4.2.9}$$

a solution of (4.2.7) exists which is unique up to $u \in \mathcal{N}(A|_U)$ *and corresponding* $\hat{u}_2 \in \widehat{H}^2_A(\Gamma)$ *such that* $u_2 - \hat{u}_2 \in \mathcal{N}(C_1)$ *where* u_2 *is the corresponding component of trace of* u *in* $\widehat{H}^2_A(\Gamma)$.

The inf-sup constant for bilinear form (4.2.8) is equal to the inf-sup constant of the adjoint operator (A^*, C_2') *from Theorem 2.* ∎

Proof. We observe that the conjugate B' of operator B corresponding to the bilinear form (4.2.8) coincides with the strong form of operator (A^*, C_2'). The result is then a direct consequence of Theorem 2. □

Ultra-weak formulation for the Stokes problem. We are now ready to discuss the ultra-weak Stokes formulation. The solution consists of $u = (\boldsymbol{u}, p, \boldsymbol{\sigma})$ and unknown traction

$$\hat{\boldsymbol{t}} = (-\boldsymbol{\sigma} + p\boldsymbol{I})\boldsymbol{n}.$$

Remember that only for a sufficiently regular[7] solution u will $\hat{\boldsymbol{t}}$ coincide with the trace $(-\boldsymbol{\sigma} + p\boldsymbol{I})\boldsymbol{n}$. In the ultra-weak formulation, the traction $\hat{\boldsymbol{t}}$ appears as an *independent* unknown. Taking the incompressibility constraint to be

[7] That is, $u \in H_A(\Omega)$.

homogeneous (that is, $g = 0$), the variational problem reads as follows:

$$\begin{cases} \text{Find } \boldsymbol{u} \in \boldsymbol{L}^2(\Omega), p \in L^2(\Omega), \boldsymbol{\sigma} \in \boldsymbol{L}^2(\Omega), \hat{\boldsymbol{t}} \in H^{-1/2}(\Gamma) \text{ such that} \\ (\boldsymbol{u}, \nabla \cdot (\boldsymbol{\tau} - q\boldsymbol{I})) \quad + (p, -\nabla \cdot q) + (\boldsymbol{\sigma}, \boldsymbol{\tau} + \nabla \boldsymbol{v}) + \langle \hat{\boldsymbol{t}}, \boldsymbol{v} \rangle \\ \qquad\qquad\qquad\qquad = (\boldsymbol{f}, \boldsymbol{v}) - \langle \boldsymbol{u}_D, (-\boldsymbol{\tau} + q\boldsymbol{I})\boldsymbol{n} \rangle \\ \qquad\qquad\qquad\qquad \forall (\boldsymbol{v}, q, \boldsymbol{\tau}) \text{ such that } \boldsymbol{\tau} - q\boldsymbol{I} \in \boldsymbol{H}(\mathrm{div}, \Omega), \boldsymbol{v} \in \boldsymbol{H}^1(\Omega). \end{cases}$$

The load is specified by a body force $\boldsymbol{f} \in \boldsymbol{L}^2(\Omega)$ and a velocity $\boldsymbol{u}_D \in \boldsymbol{H}^{1/2}(\Gamma)$ on the boundary with vanishing normal component:

$$\int_\Gamma \boldsymbol{u}_D \cdot \boldsymbol{n} = 0.$$

The solution is determined up to a constant pressure p_0 and corresponding constant traction $\hat{\boldsymbol{t}}_0 = p_0 \boldsymbol{n}$.

Notice that there are no boundary conditions imposed on the test functions. This is important from a practical point of view.

Remark 1 Strong versus weak imposition of boundary conditions. In classical variational formulations for second order PDEs, we distinguish between *strong* (Dirichlet) and *weak* (Neumann) boundary conditions. Dirichlet boundary conditions are accounted for by introducing a finite-energy lift of the boundary condition data, and solving the problem with homogeneous boundary conditions and a modified "load vector" that includes the action of the bilinear form on the lift [34, p. 34]. In practice, the Dirichlet data is first projected (interpolated) into the trace of the finite element space and then lifted with finite element shape functions. By contrast to the Dirichlet case,

Neumann conditions only contribute to the load vector. The terms "strong" and "weak" refer to the fact that, with Dirichlet data in the finite element space, Dirichlet conditions are enforced pointwise, while Neumann conditions are generally satisfied only in the limit.

In an ultra-weak variational formulation, each boundary condition may be enforced weakly or strongly. The formulation discussed above corresponds to a weak imposition; the data f_D contributes to the load vector and is accounted for on the element level, in the integration for the load vector. A strong imposition of the same condition would begin by finding a trace lift $\hat{u}_0$ of the boundary data where

$$C_1 \hat{u}_0 = f_D.$$

Notice that the lift may have a non-zero $\widehat{H}^2_A$-component but the final trace will be equal to the sum of the lift and an unknown component $\hat{u}_2 \in \widehat{H}^2_A$,

$$\hat{u} = \hat{u}_0 + \hat{u}_2.$$

The term $\langle f_D, v \rangle$ on the right-hand side of (4.2.8) is simply replaced with $c(u_0, v)$. The rest of the formulation remains unchanged. The difference between the two formulations will become clearer in the context of discontinuous test functions discussed in the next section. ∎

4.3 DPG formulation

The essence of the DPG formulation lies in extending the concept of the ultra-weak variational formulation to broken test spaces. We begin by

partitioning domain Ω into finite elements K and integrating by parts on each element:

$$(Au, v)_K = (u, A^* v)_K + c_{\partial K}(u, v) \quad u \in H_A(K), v \in H_{A^*}(K).$$

Next, we sum over all elements to obtain

$$\underbrace{\sum_K (Au, v)_K}_{=(Au,v)} = \underbrace{\sum_K (u, A^* v)_K}_{=:(u, A_h^* v)_h} + \underbrace{\sum_K c_{\partial K}(u, v)}_{=:c_h(u,v)} \quad u \in H_A(\Omega), v \in H_{A^*}(\Omega_h).$$

Here we take u to be globally conforming but allow v to come from the broken graph space

$$H_{A^*}(\Omega_h) := \left\{ v \in L^2(\Omega) \ : \ A^* v|_K \in L^2(K) \, \forall K \right\}.$$

The index h on the domain indicates that the formal adjoint operator is to be understood element-wise. The boundary term c_h now extends to the whole skeleton $\Gamma_h = \cup_K \partial K$. For the internal skeleton $\Gamma_h^0 = \Gamma_h - \Gamma$, this term represents the action of traces $\hat{u}$ on the jumps of traces $\hat{v}$. Recalling our definitions of $H^{1/2}(\Gamma_h)$ and $H^{-1/2}(\Gamma_h)$ in Section 4.1, we analogously introduce a general, abstract space of traces on the skeleton,

$$\widehat{H}_A(\Gamma_h) := \left\{ \hat{u} = \{\hat{u}_K\} \in \prod_K \widehat{H}_A(\partial K) \ : \ \exists u \in H_A(\Omega) : \mathrm{tr}_A u|_K = \hat{u}_K \right\},$$

and the corresponding subspace of traces that vanish on $\Gamma = \partial\Omega$,

$$\hat{\widehat{H}}_A(\Gamma_h) := \left\{ \hat{u} = \{\hat{u}_K\} \in \prod_K \widehat{H}_A(\partial K) \ : \ \exists u \in \tilde{H}_A(\Omega) : \mathrm{tr}_A u|_K = \hat{u}_K \right\},$$

where

$$\tilde{H}_A(\Omega) = \{u \in H_A(\Omega) \,:\, \text{tr } u = 0 \text{ on } \Gamma\}\,.$$

As usual, we equip the trace space with the minimum energy extension norm.

Any function $u \in H_A(\Omega)$ can be decomposed into an extension of its trace to Γ and a component that vanishes on Γ:

$$u = E(\text{tr } u) + \tilde{u}, \quad \tilde{u} \in \tilde{H}_A(\Omega).$$

If the extension $E(\text{tr } u)$ is the minimum-energy extension, the decomposition above is H_A-orthogonal. This implies a corresponding decomposition for traces $\hat{u} \in \widehat{H}_A(\Gamma_h)$:

$$\hat{u} = \hat{E}\hat{u}_0 + \hat{\tilde{u}}, \quad \hat{\tilde{u}} \in \hat{\tilde{H}}_A(\Gamma_h).$$

Here $\hat{u}_0 \in \widehat{H}_A(\Gamma)$ is the restriction of $\hat{u}$ to Γ and $\hat{E}\hat{u}_0 \in \widehat{H}_A(\Gamma_h)$ is any extension of $\hat{u}_0$ back to the whole skeleton Γ_h. Again, if we use the minimum-energy extension, the decomposition is $\widehat{H}_A(\Gamma_h)$-orthogonal. We have

$$\widehat{H}_A(\Gamma_h) = \hat{E}\widehat{H}_A(\Gamma) \oplus \hat{\tilde{H}}_A(\Gamma_h). \tag{4.3.10}$$

By construction, we have a generalization of the trace operator to the whole skeleton,

$$\text{tr} : H_A(\Omega) \twoheadrightarrow \widehat{H}_A(\Gamma_h).$$

The skeleton term $c_h(\hat{u}, \hat{v})$ is well-defined for $\hat{u} \in \widehat{H}_A(\Gamma_h)$ and $\hat{v} = \{\hat{v}_K\} \in \prod_K H_{A^*}(\partial K)$. We also have the condition

$$\Big((c_h(\hat{u}, \hat{v}) = 0 \,\forall \hat{u} \in \hat{\tilde{H}}_A(\Gamma_h)\Big) \Longleftrightarrow v \in H_{A^*}(\Omega).$$

That is, if we restrict ourselves to globally conforming test functions, the skeleton term reduces to a term that involves only the domain boundary Γ, where the trace $\hat{u}$ vanishes. The converse follows from the definition of distributional derivatives. For any test function $\phi \in \mathcal{D}(\Omega)$, we have

$$c_h(\mathrm{tr}\phi, v) = (A\phi, v) - (\phi, A_h^* v)_h = 0,$$

which proves that the union of element-wise values $A_h^* v$ (which lives in $L^2(\Omega)$) is equal to $A^* v$ in the sense of distributions.

We now use the decomposition of traces (4.3.10) to set up the boundary operators. Recall that condition (4.2.4) decomposed the trace space $\widehat{H}_A(\Gamma)$ into the direct sum of the nullspaces of operators C_2 and C_1:

$$\widehat{H}_A(\Gamma) = \widehat{H}_A^1(\Gamma) \oplus \widehat{H}_A^2(\Gamma), \quad \hat{u} = \hat{u}_1 + \hat{u}_2.$$

The first term is known from the boundary condition; the second remains as an additional unknown. We have

$$\begin{aligned} c_h(\hat{u}, \hat{v}) &= c_h(\hat{E}\hat{u}_0, \hat{v}) + c_h(\hat{\tilde{u}}, \hat{v}) \\ &= c_h(\hat{E}\hat{u}_0^1, \hat{v}) + c_h(\hat{E}\hat{u}_0^2, \hat{v}) + c_h(\hat{\tilde{u}}, \hat{v}). \end{aligned} \tag{4.3.11}$$

For conforming test functions $\hat{v} \in \widehat{H}_{A^*}(\Gamma_h)$, the bilinear form on the skeleton reduces to the bilinear form on the domain boundary,

$$c_h(\hat{u}, \hat{v}) = c(\hat{u}, \hat{v}) = c(\hat{u}_0^1, \hat{v}) + c(\hat{u}_0^2, \hat{v}) = \langle f_D, \hat{v} \rangle + c(\hat{u}_0^2, \hat{v}).$$

In particular, this is independent of the choice of lift $\hat{E}$. To impose the boundary condition strongly, we require a trace lift $\hat{u}_0$ of the boundary data f_D such that

$$C_1 \hat{u}_0 = f_D.$$

We then move the term with the lift to the right-hand side, and keep as an unknown the component $\hat{u}_2$ of the trace on Γ. The final formulation reads as follows:

$$\begin{cases} u \in L^2(\Omega), \hat{u} \in \hat{E}\widehat{H}^2_A(\Gamma) \oplus \hat{\widehat{H}}_A(\Gamma_h), \\ (u, A_h^* v)_h + c_h(\hat{u}, v) = (f, v) - c_h(\hat{E}\hat{u}_0, v), \quad \forall v \in H_{A^*}(\Omega_h). \end{cases}$$

If we instead enforce the boundary condition weakly, we need to replace the first term on the right-hand side of (4.3.11) with an extension of known boundary data $\langle f_D, v\rangle$ to discontinuous test functions. This is always possible as the term $c_h(\hat{E}\hat{u}_0, v)$ provides an example of such an extension.

The final abstract DPG formulation[8] is then

$$\begin{cases} u \in L^2(\Omega), \hat{u} \in \hat{E}\widehat{H}^2_A(\Gamma) \oplus \hat{\widehat{H}}_A(\Gamma_h), \\ (u, A_h^* v)_h + c_h(\hat{u}, v) = (f, v) - \langle f_D, v\rangle_{\Gamma_h} \quad \forall v \in H_{A^*}(\Omega_h). \end{cases} \tag{4.3.12}$$

The bilinear[9] form corresponding to the formulation

$$b((u, \hat{u}), v) := (u, A_h^* v)_h + c_h(\hat{u}, v)$$

generates operators B and B', where

$$b((u, \hat{u}), v) = \langle B(u, \hat{u}), v\rangle = \langle (u, \hat{u}), B'v\rangle.$$

The null space of conjugate operator B' coincides with the null space of $A^*|_V$. Indeed, let

$$b((u, \hat{u}), v) = 0 \quad \forall (u, \hat{u}).$$

[8]That is, the ultra-weak variational formulation with broken test functions.
[9]Sesquilinear for complex-valued problems.

Taking arbitrary $\hat{u} \in \hat{\tilde{H}}_A(\Gamma_h)$, we conclude that v must be globally conforming, so the bilinear form in (4.3) reduces to the bilinear form (4.2.8).

The null space of DPG operator B consists of all $(u, \hat{u})$ such that

$$b((u, \hat{u}), v) = 0 \quad \forall v \in H_{A^*}(\Omega_h).$$

As with the ultra-weak variational formulation, we first test with $v \in \mathcal{D}(\Omega)$ to conclude that $Au = 0$. Integrating the first term by parts and testing with arbitrary v, we conclude that $\hat{u} = u$ on Γ_h. In particular, as $\hat{u}|_\Gamma \in \widehat{H}^2_A(\Gamma)$, this implies that $C_1 u = 0$ on Γ.

We are in a position to state our main abstract result.

THEOREM 4

Problem (4.3.12) is well-posed. More precisely, for any data f and f_D that satisfy the compatibility condition (4.2.9), the problem has a solution $(u, \hat{u})$ such that $(u, \hat{u}|_\Gamma)$ coincides with the solution of (4.2.7). The bilinear form satisfies the inf-sup condition

$$\sup_{v \in H_{A^*}(\Omega_h)} \frac{|(u, A_h^* v)_h + c_h(\hat{u}, v)|}{\|v\|_{H_{A^*}(\Omega_h)}} \geq \gamma_{DPG} \left(\|u\|^2_{L^2(\Omega)} + \|\hat{u}\|^2_{\widehat{H}_A(\Gamma_h)} \right)^{1/2}$$

for all $\hat{u} \in \hat{E}\widehat{H}^2_A(\Gamma) \oplus \hat{\tilde{H}}_A(\Gamma_h)$, and $u \in L^2(\Omega)$ orthogonal to the null space:

$$\{(u, \hat{u}) \ : \ u \in \mathcal{N}(A|_U) \text{ and } \hat{u} = u \text{ on } \Gamma_h\}$$

The inf-sup constant γ_{DPG} is mesh-independent and $\gamma_{DPG} = O(\gamma)$ and $O(\tilde{\gamma})$ for the adjoint operator. ∎

Proof. We will switch the order of spaces in the inf-sup condition[10] and prove that

$$\sup_{\substack{u\in L^2(\Omega)\\ \hat{u}\in E\widehat{H}^2_A(\Gamma)\oplus\hat{\tilde{H}}_A(\Gamma_h)}} \frac{|(u,A_h^*v)_h + c_h(\hat{u},v)|}{\left(\|u\|^2_{L^2(\Omega)} + \|\hat{u}\|^2_{\widehat{H}_A(\Gamma_h)}\right)^{1/2}} \geq \gamma_{DPG}\|v\|_{H_{A^*}(\Omega_h)} \tag{4.3.13}$$

for all v L^2-orthogonal to $\mathcal{N}(A^*|_V)$.

Step 1: Let us first consider the special case when $A_h^*v = 0$. Take a conforming $u \in (\mathcal{N}(A|_U))^\perp \subset U$ such that $Au = v$; since $v \in (\mathcal{N}(A^*|_V))^\perp$, such a u exists. We then have

$$\begin{aligned}
\|v\|^2 &= (Au,v) = \underbrace{(u,A_h^*v)}_{=0} + c_h(\mathrm{tr}_A\, u, v)\\
&\leq \frac{|c_h(\mathrm{tr}_A\, u, v)|}{\|\mathrm{tr}_A\, u\|}\, \|\mathrm{tr}_A\, u\|\\
&\leq \sup_{\hat{u}} \frac{|c_h(\hat{u},v)|}{\|\hat{u}\|}\, \|u\|_{H_A(\Omega)}\\
&\leq \frac{1}{\gamma}\sup_{\hat{u}} \frac{|c_h(\hat{u},v)|}{\|\hat{u}\|}\|v\|.
\end{aligned}$$

Dividing both sides by $\|v\|$, we get the required inequality.

Step 2: Now let v be arbitrary. Consider a conforming $\tilde{v} \in H_{A^*}(\Omega)$ such that $A^*\tilde{v} = A_h^*v$. By Assumption 1, such a function always exists and can be interpreted as a solution to the strong adjoint problem with non-

[10] In general, one may switch the order of the spaces in the inf-sup condition if the operator and its adjoint are both injective (see [41] for details). When the operator and its adjoint are not injective, we *make* them injective by considering quotient spaces instead (dividing through by the null space). In the Hilbert space setting, quotient spaces are isomorphic and isometric to orthogonal complements, our present setting.

homogeneous BC data $f_D = C_2'\tilde{v}$:

$$\begin{cases} A^*\tilde{v} = A_h^* v \\ C_2'\tilde{v} = f_D. \end{cases}$$

To ensure uniqueness and boundedness in the L^2-norm, we assume that $\tilde{v}$ is L^2-orthogonal to the null space $\mathcal{N}(A^*|_V)$.

Now, by construction, $A_h(v - \tilde{v}) = 0$ and $v - \tilde{v} \in (\mathcal{N}(A^*|_V))^\perp$ so that, by the Step 1 result, the difference $v - \tilde{v}$ is bounded in both L^2 and H_{A^*} norms by the supremum in (4.3.13). We thus need only demonstrate that we can control the norm of the conforming $\tilde{v}$. But, if we restrict ourselves in (4.3.13) to conforming test functions, the bilinear form collapses to (4.2.8).

This completes the proof. □

4.4 DPG formulation for the Stokes problem

We begin by emphasizing the global character of the decomposition of traces in (4.3.10). The velocity trace $\hat{\boldsymbol{u}} \in \boldsymbol{H}^{1/2}(\Gamma_h)$ can be decomposed into an extension of $\hat{\boldsymbol{u}}_0$ on the boundary Γ and the trace on the internal skeleton Γ_h^0:

$$\hat{\boldsymbol{u}} = \hat{E}\hat{\boldsymbol{u}}_0 + \hat{\tilde{\boldsymbol{u}}}.$$

In finite element computations, traces are approximated with functions that are globally continuous on the skeleton Γ_h. The trace $\hat{\boldsymbol{u}}_0$, which is known from the boundary condition, has to be lifted to the whole skeleton. In computations, we use finite element shape functions and lift $\hat{\boldsymbol{u}}_0$ only into

the layer of elements neighboring Γ. The unknown part of velocity trace $\hat{\tilde{\boldsymbol{u}}} \in \tilde{\boldsymbol{H}}^{1/2}(\Gamma_h^0)$, by contrast, is defined only on the internal skeleton.

The unknown traction trace $\hat{\boldsymbol{t}} \in \boldsymbol{H}^{-1/2}(\Gamma_h)$ is defined on the entire skeleton. On the continuous level, the decomposition of traction into a lift of its restriction to Γ and the remaining component $\hat{\tilde{\boldsymbol{t}}} \in \tilde{\boldsymbol{H}}^{-1/2}(\Gamma)$ defined on the internal skeleton Γ_h^0 is also global. In 2D for instance, for $\boldsymbol{t}$ from a *standard* boundary space $\boldsymbol{H}^{-1/2}(\Gamma)$, the corresponding restriction to an edge e of an element K adjacent to boundary Γ lives only in $\boldsymbol{H}^{-1/2}(e)$ and *cannot* just be extended by zero to a functional in $\boldsymbol{H}^{-1/2}(\partial K)$. However, the conformity present in the definition of space $\boldsymbol{H}^{-1/2}(\Gamma_h)$ is so weak that it does not translate into any global continuity conditions for the approximating polynomial spaces that are discontinuous from edge to edge.

For the Stokes problem, the boundary operators represent exactly the velocity and traction components of the solution trace;

$$C_1(\hat{\boldsymbol{t}}, \hat{\boldsymbol{u}}) = \boldsymbol{u}, \quad C_2(\hat{\boldsymbol{t}}, \hat{\boldsymbol{u}}) = \hat{\boldsymbol{t}}.$$

The difference between the strong and weak imposition of boundary conditions is, in our case, insignificant. The abstract $\langle f_D, v\rangle$ term corresponds to $\langle \boldsymbol{u}_D, \boldsymbol{r}\rangle_\Gamma$, where $\boldsymbol{r}$ is the traction component of the test function. Its extension to discontinuous test functions v is constructed by lifting Dirichlet data $\boldsymbol{u}_D$ to the whole skeleton. The strong imposition of the boundary conditions is essentially the same: the abstract lift $\hat{u}_0$ of $\boldsymbol{u}_D$ can be selected to be $(\boldsymbol{u}_D, \boldsymbol{0})$ (zero traction) and, if we use the same extension of $\boldsymbol{u}_D$ to the whole skeleton,

the two formulations will be identical. A subtle difference lies in the way we treat those lifts in finite element computations. Suppose, for example, that we approximate traces with quadratics and have non-polynomial data u_D. If we impose boundary conditions strongly, we first interpolate the data with quadratics and use quadratic shape functions to lift it to the whole skeleton. The contributions to the load vector will be computed by integrating the quadratic lifts against test functions. If we instead impose boundary conditions weakly, the non-polynomial $\boldsymbol{u}_D$ on Γ will be integrated directly against the test functions and, in general, will yield different values. Additionally, even if we lift the non-polynomial $\boldsymbol{u}_D$ to the whole skeleton with the same quadratic shape functions, the lifts will differ on the internal skeleton and, consequently, the resulting approximate traces will differ. In the limit, of course, the difference will disappear.

The null space of conjugate operator B' coincides with the null space of adjoint A^* with homogeneous boundary conditions $\boldsymbol{v} = \mathbf{0}$ on Γ and consists of constant pressures

$$\{(\mathbf{0}, c, \mathbf{0}) \ : \ c \in \mathbb{R}\}.$$

The null space of operator B is the same as that for the operator corresponding to the ultra-weak formulation,

$$\{((\boldsymbol{u}, p, \boldsymbol{\sigma}), \hat{\boldsymbol{t}}) \ : \ \boldsymbol{u} = \mathbf{0}, p = c, \boldsymbol{\sigma} = \mathbf{0}, \hat{\boldsymbol{t}} = c\boldsymbol{n} \text{ where } c \in \mathbb{R}\}.$$

The non-trivial null spaces imply the compatibility condition for the load and non-uniqueness of the solution. The compatibility condition for the load

involves the right-hand side g of the divergence equation[11] and the velocity trace boundary data $\boldsymbol{u}_D$, and takes the form

$$\int_\Omega g = \int_\Gamma \boldsymbol{u}_D \cdot \boldsymbol{n}.$$

This well-known condition can be obtained immediately by integrating the divergence equation and using the boundary condition on $\boldsymbol{u}$:

$$\int_\Omega g = \int_\Omega \operatorname{div} \boldsymbol{u} = \int_\Gamma \boldsymbol{u} \cdot \boldsymbol{n} = \int_\Gamma \boldsymbol{u}_D \cdot \boldsymbol{n}.$$

Assumption 1 thus reduces to the condition that the divergence operator is surjective, a well-known fact.

With data satisfying the compatibility condition, the solution (pressure and tractions) is determined up to a constant. In computations, the constant can be fixed by implementing an additional scaling condition. We can enforce, for instance, zero average pressure in one particular element, or zero average normal traction on a particular edge. The scaling will affect the ultimate values for pressure and tractions, but has no effect on the velocity or on its gradient and trace.

4.5 A summary

We have presented a general theory for DPG variational formulations, making a number of assumptions on the first-order, linear differential operator A, which we now summarize.

[11] $g = 0$ in practice.

- Operator A and its formal adjoint A^* are surjective (Assumption 1).
- Both energy graph spaces $H_A(\Omega)$ and $H_{A^*}(\Omega)$ admit corresponding trace spaces $\widehat{H}_A(\partial\Omega), \widehat{H}_{A^*}(\partial\Omega)$.
- The boundary bilinear term $c(u, v)$ resulting from integration by parts is definite.
- Boundary operator C_1 has been selected in such a way that Assumption 2 is satisfied.
- With homogeneous boundary condition $C_1 u = 0$ in place, operator A is bounded below in the L^2-orthogonal complement of its null space (Assumption 3).

With these conditions satisfied, the DPG formulation is well-posed. The corresponding inf-sup constant is mesh-independent. Neglecting technical details, the central message is this: *the boundedness below of the strong operator with homogeneous boundary conditions implies the inf-sup condition for the DPG formulation with a mesh-independent constant.*

The general theory guides our definition of unknown traces on the skeleton. The energy setting involves graph norms for both operator A and its formal adjoint A^*. The graph norm on the test space is equivalent to the *optimal test norm* [78] with *mesh-independent* equivalence constants. The graph norm for A determines the energy setting for unknown traces and the minimum energy extension norm.

All these conditions are satisfied for the Stokes problem.

4.6 Boundedness Below of the First-Order Stokes Operator with Homogeneous BCs

Finally, for completeness we show that the operator corresponding to our first-order Stokes problem with homogeneous BCs is bounded below. While in the above we have assumed non-dimensionalization so that $\mu = 1$, here we consider the case of constant $\mu > 0$.

Recall the classical strong form of the Stokes problem:

$$-\mu\Delta\boldsymbol{u} + \nabla p = \boldsymbol{f} \qquad \text{in } \Omega, \tag{4.6.14}$$

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{in } \Omega, \tag{4.6.15}$$

$$\boldsymbol{u} = \boldsymbol{u}_D \qquad \text{on } \partial\Omega. \tag{4.6.16}$$

Recall also our first-order system for the Stokes equations:

$$-\nabla \cdot \boldsymbol{\sigma} + \nabla p = \boldsymbol{f} \qquad \text{in } \Omega, \tag{4.6.17a}$$

$$\nabla \cdot \boldsymbol{u} = \boldsymbol{0} \qquad \text{in } \Omega, \tag{4.6.17b}$$

$$\frac{1}{\mu}\boldsymbol{\sigma} - \nabla\boldsymbol{u} = 0 \qquad \text{in } \Omega, \tag{4.6.17c}$$

$$\boldsymbol{u} = \boldsymbol{u}_D \qquad \text{on } \partial\Omega. \tag{4.6.17d}$$

If we introduce $A : H_A \to \boldsymbol{L}^2$ with group variable $u = (\boldsymbol{u}, p, \boldsymbol{\sigma})$, the Stokes equation in first order form (4.6.17), ignoring the boundary conditions,

can be written succinctly as

$$Au \stackrel{\text{def}}{=} \begin{bmatrix} -\nabla \cdot \boldsymbol{\sigma} + \nabla p \\ \nabla \cdot \boldsymbol{u} \\ \frac{1}{\mu}\boldsymbol{\sigma} - \nabla \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ 0 \\ \mathbf{0} \end{bmatrix}. \tag{4.6.18}$$

Considering linear operator A as defined in equation (4.6.18) operating on group variable $u = (\boldsymbol{u}, p, \boldsymbol{\sigma})$, we seek to show that $||Au||_{L^2} \geq \gamma ||u||_{H_A}$. Adding right-hand sides g and $\boldsymbol{h}$

$$-\nabla \cdot \boldsymbol{\sigma} + \nabla p = \boldsymbol{f} \qquad \text{in } \Omega, \tag{4.6.19}$$

$$\nabla \cdot \boldsymbol{u} = g \qquad \text{in } \Omega, \tag{4.6.20}$$

$$\frac{1}{\mu}\boldsymbol{\sigma} - \nabla \boldsymbol{u} = \boldsymbol{h} \qquad \text{in } \Omega, \tag{4.6.21}$$

$$\boldsymbol{u} = \mathbf{0} \qquad \text{on } \partial\Omega, \tag{4.6.22}$$

we have that $Au = (\boldsymbol{f}, g, \boldsymbol{h})$. If we can establish bounds for the L^2 norms of each of the solution variables in terms of L^2 norms on $\boldsymbol{f}, g$, and $\boldsymbol{h}$, then we will have the required lower bound $||Au||_\Omega \geq \gamma ||u||_{H_A}$, where $||u||_{H_A} \stackrel{\text{def}}{=} (||Au||_\Omega^2 + ||u||_\Omega^2)^{1/2}$.

Note that, by linearity of A, it suffices to consider cases in which only one of $\boldsymbol{f}, g$, and $\boldsymbol{h}$ is non-zero. We consider each of these cases in turn.

$\boldsymbol{f} \neq \mathbf{0}, g = 0, \boldsymbol{h} = \mathbf{0}$. In this case, we have exactly the system (4.6.17a)-(4.6.17d), which reduces (in a distributional sense) to (4.6.14)-(4.6.16). Testing the first equation with the velocity $\boldsymbol{u} \in \boldsymbol{H}_0^1(\Omega)$, we have

$$-(\mu \Delta \boldsymbol{u}, \boldsymbol{u})_\Omega + (\nabla p, \boldsymbol{u})_\Omega = (\boldsymbol{f}, \boldsymbol{u})_\Omega.$$

Integrating by parts, we obtain:

$$(\mu\nabla\boldsymbol{u},\nabla\boldsymbol{u})_\Omega - \langle\mu\nabla\boldsymbol{u}\cdot\boldsymbol{n},\boldsymbol{u}\rangle_{\partial\Omega} - (p,\nabla\cdot u)_\Omega + \langle p,\boldsymbol{u}\cdot\boldsymbol{n}\rangle_{\partial\Omega} = (\boldsymbol{f},\boldsymbol{u})_\Omega.$$

Noting that $\boldsymbol{u}=0$ on $\partial\Omega$ and that $\nabla\cdot\boldsymbol{u}=0$ in Ω, this reduces to:

$$\begin{aligned}\mu(\nabla\boldsymbol{u},\nabla\boldsymbol{u})_\Omega = \mu||\nabla\boldsymbol{u}||_\Omega^2 = \mu||\boldsymbol{\sigma}||_\Omega^2 &= (\boldsymbol{f},\boldsymbol{u})_\Omega \\ &\leq ||f||_\Omega||\boldsymbol{u}||_\Omega \\ &\leq C_P||f||_\Omega||\nabla\boldsymbol{u}||_\Omega,\end{aligned}$$

where C_P is the Poincaré constant. Thus $||\boldsymbol{\sigma}||_\Omega \leq \frac{C_P}{\mu}||f||_\Omega$, and $||\boldsymbol{u}|| \leq \frac{C_P^2}{\mu}||f||_\Omega$.

To bound the pressure p, we require the inf-sup condition (which holds for Lipschitz as well as more general domains, as discussed in Section 2.2 of [32]) for $p \in L_0^2 \stackrel{\text{def}}{=} \{w \in L^2(\Omega) : \int_\Omega w = 0\}$:

$$\sup_{\boldsymbol{v}\in\boldsymbol{H}^1(\Omega)} \frac{(p,\nabla\cdot\boldsymbol{v})_\Omega}{\|\boldsymbol{v}\|_{H^1(\Omega)}} \geq \beta\,\|p\|_\Omega \tag{4.6.23}$$

for some constant $\beta > 0$. Testing equation (4.6.14) with $\boldsymbol{v} \in \boldsymbol{H}_0^1(\Omega)$, we have

$$(\nabla p,\boldsymbol{v})_\Omega = (\boldsymbol{f},\boldsymbol{v})_\Omega + (\mu\Delta\boldsymbol{u},\boldsymbol{v})_\Omega.$$

Integrating by parts, dividing by $\|\boldsymbol{v}\|_{H^1(\Omega)}$ and taking the supremum:

$$\begin{aligned}\sup -\frac{(p,\nabla\cdot\boldsymbol{v})}{\|\boldsymbol{v}\|_{H^1(\Omega)}} &= \sup\left\{\frac{(\boldsymbol{f},\boldsymbol{v})}{\|\boldsymbol{v}\|_{H^1(\Omega)}} - \frac{\mu(\nabla\boldsymbol{u},\nabla\boldsymbol{v})}{\|\boldsymbol{v}\|_{H^1(\Omega)}}\right\} \\ &\leq \frac{\|\boldsymbol{f}\|\,\|\boldsymbol{v}\|_{L^2}}{\|\boldsymbol{v}\|_{H^1(\Omega)}} + \mu\,\|\nabla\boldsymbol{u}\| \leq \|\boldsymbol{f}\| + \mu\frac{C_P}{\mu}\,\|\boldsymbol{f}\| = (1+C_P)\,\|\boldsymbol{f}\|.\end{aligned}$$

By the inf-sup condition, we then have $\|p\| \leq \frac{1+C_P}{\beta} \|\boldsymbol{f}\|$, bounding p, as required.

$\boldsymbol{f} = \boldsymbol{0}, g = 0, \boldsymbol{h} \neq \boldsymbol{0}$. In this case, we have

$$-\nabla \cdot \boldsymbol{\sigma} + \nabla p = \boldsymbol{0} \qquad \text{in } \Omega, \tag{4.6.24}$$

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{in } \Omega, \tag{4.6.25}$$

$$\frac{1}{\mu}\boldsymbol{\sigma} - \nabla \boldsymbol{u} = \boldsymbol{h} \qquad \text{in } \Omega, \tag{4.6.26}$$

$$\boldsymbol{u} = \boldsymbol{0} \qquad \text{on } \partial\Omega. \tag{4.6.27}$$

Then, $\boldsymbol{\sigma} = \mu(\nabla \boldsymbol{u} + \boldsymbol{h})$, and we have:

$$-\nabla \cdot \boldsymbol{\sigma} + \nabla p = -\mu \Delta \boldsymbol{u} - \mu \nabla \cdot \boldsymbol{h} + \nabla p = \boldsymbol{0},$$

so that

$$-\mu \Delta \boldsymbol{u} + \nabla p = -\mu \nabla \cdot \boldsymbol{h}$$

in a distributional sense. Testing with $\boldsymbol{u}$, and integrating the left hand side by parts, again the pressure term vanishes because $\nabla \cdot \boldsymbol{u} = 0$, so that much as before, we obtain:

$$\begin{aligned} \mu(\nabla \boldsymbol{u}, \nabla \boldsymbol{u})_\Omega &= \mu(\nabla \cdot \boldsymbol{h}, \boldsymbol{u})_\Omega \\ &= -\mu(\boldsymbol{h}, \nabla \boldsymbol{u})_\Omega \leq \mu \|\boldsymbol{h}\|_{L^2(\Omega)} \|\nabla \boldsymbol{u}\|_{L^2(\Omega)}. \end{aligned}$$

So $\|\nabla \boldsymbol{u}\| \leq \|\boldsymbol{h}\|$, and the bounds $\|\boldsymbol{u}\| \leq C_P \|\boldsymbol{h}\|$ and $\|p\| \leq \frac{\mu}{\beta}(\|\nabla \boldsymbol{u}\| + \|\boldsymbol{h}\|) \leq \frac{2\mu}{\beta}||\boldsymbol{h}||$ can be established in a similar fashion as above.

$\boldsymbol{f} = \mathbf{0}, g \neq 0, \boldsymbol{h} = \mathbf{0}$. In this case, we have

$$-\nabla \cdot \boldsymbol{\sigma} + \nabla p = \mathbf{0} \qquad \text{in } \Omega, \tag{4.6.28}$$

$$\nabla \cdot \boldsymbol{u} = g \qquad \text{in } \Omega, \tag{4.6.29}$$

$$\frac{1}{\mu}\boldsymbol{\sigma} - \nabla \boldsymbol{u} = 0 \qquad \text{in } \Omega, \tag{4.6.30}$$

$$\boldsymbol{u} = \mathbf{0} \qquad \text{on } \partial\Omega. \tag{4.6.31}$$

We must also assume compatibility between g and the boundary condition on $\boldsymbol{u}$; that is, that g has zero average on Ω:

$$\int_\Omega g = \int_{\partial\Omega} \boldsymbol{u} \cdot \boldsymbol{n} = 0.$$

The inf-sup condition (4.6.23) is equivalent[12] to the existence, for every $g \in L^2(\Omega)$, of $\boldsymbol{u}_0 \in H_0^1$ such that

$$\nabla \cdot \boldsymbol{u}_0 = g \quad \text{and} \quad \|\boldsymbol{u}_0\| \leq C \|g\|.$$

Defining $\boldsymbol{w} \in H_0^1$ by $\boldsymbol{w} = \boldsymbol{u} - \boldsymbol{u}_0$, we substitute $\boldsymbol{u} = \boldsymbol{w} + \boldsymbol{u}_0$ into the system (4.6.28)-(4.6.31), obtaining

$$-\nabla \cdot \boldsymbol{\sigma} + \nabla p = \mathbf{0} \qquad \text{in } \Omega,$$

$$\nabla \cdot \boldsymbol{w} = \nabla \cdot (\boldsymbol{u} - \boldsymbol{u}_0) = 0 \qquad \text{in } \Omega,$$

$$\frac{1}{\mu}\boldsymbol{\sigma} - \nabla \boldsymbol{w} = \nabla \boldsymbol{u}_0 \qquad \text{in } \Omega,$$

$$\boldsymbol{w} = \mathbf{0} \qquad \text{on } \partial\Omega.$$

Thus we have reduced the $g \neq 0$ case to the $\boldsymbol{h} \neq 0$ case. This completes the proof.

[12] Again, see Section 2.2 of [32].

Chapter 5

Verification of Camellia

We adopt a multi-pronged approach to verification of the code. The first line of defense is runtime parameter checking, some of which is provided by Trilinos (e.g. the `FieldContainer` multi-dimensional array class checks the rank and bounds of its arguments), and some by Camellia itself (e.g. summing `LinearTermPtr`s of unlike type—say, a test and a field variable—causes an exception to be thrown). We have a growing suite of unit tests which we periodically run during code development. We have run a variety of tests to confirm expected convergence rates. Finally, we use the code in a variety of research applications—we have used it to solve Poisson, convection-dominated diffusion, Stokes, Burgers, and compressible Navier-Stokes equations; the success of each of these applications acts as one more check of the code's correctness.

We have performed a host of verification tests using Camellia to solve the Stokes problem using the velocity-stress-pressure (VSP) and velocity-vorticity-pressure (VVP) formulations, as well as the Poisson problem, on a variety of mesh types (including quadrilateral, triangular, and mixed quadrilateral and triangular). The upshot of these results is that we do

converge at the expected rates for these problems; full details can be found in Appendix B.

In this chapter, we focus our attention on several tests of the curvilinear geometry. We choose these for two reasons. First, they exemplify our general approach to testing: we begin with unit-level tests that confirm that individual methods produce expected results, then turn to tests of progressively higher-level code, culminating in tests that mirror the kinds of computations we use Camellia to study. In this case, we build up to a study of convergence rates in the solution to the Poisson problem on a curvilinear mesh. The second reason we focus on curvilinear geometry is that this latter study corroborates our hypothesis that refinements whose interior edges are straight suffice[1] in the context of curvilinear DPG to achieve optimal convergence rates—and this is the approach we employ in Chapter 8 when solving the flow past a cylinder problem using the Navier-Stokes equations.

5.1 Unit Tests

As a sanity check on our curvilinear Jacobian computation, we create a curvilinear mesh whose curved edges are simply straight lines and a standard straight-edge mesh (i.e. not curvilinear from Camellia's point of view), and compare the computed Jacobians for a variety of polynomial orders and mesh

[1] As suggested in Chapter 2, this is subject to certain constraints on the geometry: the straight edges must of course remain in the interior of the elements. For the initial meshes and the isotropic refinements that we employ here and in Chapter 8, it is clear that this requirement will be satisfied.

widths, confirming that these are identical. A more involved test checks various details of the H^1 projection-based interpolation, confirming that the geometry basis coefficients match analytically determined values, that the projection recovers geometry exactly representable in the space, and so on.

5.2 Area of a Straight-Edged Mesh

We run two simple verification tests. The first simply replaces a single-element straight-edged mesh by one with "curvilinear" edges, but these edges are in fact parametrically specified straight lines, so that the straight-edged and physical mesh should be identical. We then verify, for varying polynomial orders, that the area of the mesh is correctly computed.

5.3 Implied value of π

The next test uses a mesh generated by `MeshFactory::hemkerMesh()`—essentially a rectangle with a circle cut out—and, by computing its area, determines an implied value of π for meshes of varying polynomial orders of approximation. Similarly, we refine in h with a mesh of quadratic elements. The results are reported in Tables 5.1 and 5.2. Some plots of the meshes can be seen in Figures 5.3 and 5.2.

k	Implied π value	Error
1	2.82842712474621	3.13e-01
2	3.13835721343435	3.24e-03
3	3.14159111784363	1.54e-06
4	3.14159265122339	2.37e-09
5	3.14159265354021	4.96e-11

Table 5.1: Convergence in k of the value of π implied by the computation of area on our approximate geometry. Each mesh contains 20 elements.

Refinements	Implied π value	Error
0	3.13835721343435	3.24e-03
1	3.14138648117819	2.06e-04
2	3.14157970417193	1.29e-05
3	3.14159184324888	8.10e-07
4	3.14159260292922	5.07e-08
5	3.14159265042656	3.16e-09

Table 5.2: Convergence in h of the value of π implied by the computation of area on our approximate geometry. The starting mesh has 20 quadratic elements, and is uniformly refined.

5.4 Integration Test: Poisson Manufactured Solution

As a final, integration test, we consider the Poisson problem with exact solution $\phi_{\text{exact}}(x, y) = \sin x \sin y$ on the domain $(-5, 5) \times (-5, 5)$ with the unit

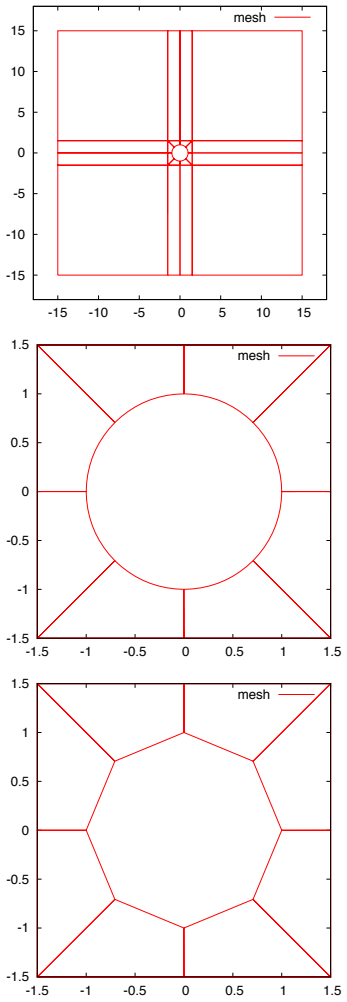


Figure 5.1: Starting mesh for the Poisson curvilinear test problem: computational geometry for full mesh (top), detail around the cylinder (middle), and linear approximation for the detail (bottom).

circle centered at the origin deleted. Taking $\psi = \nabla\phi$ and defining trace $\widehat{\phi}$ and flux $\widehat{\psi}_n$, the variational formulation is

$$\begin{aligned} b((\phi, \boldsymbol{\psi}, \widehat{\phi}, \widehat{\psi}_n, (\boldsymbol{q}, v)) = &- \int_\Omega \phi \nabla \cdot \mathbf{q} - \int_{\mathbf{\Omega}} \boldsymbol{\psi} \cdot \mathbf{q} + \int_{\partial\mathbf{\Omega}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n} \\ &- \int_\Omega \boldsymbol{\psi} \cdot \nabla v + \int_\Omega \widehat{\psi}_n\, v \;= \int_\Omega f v, \end{aligned}$$

where $f = \Delta\phi_{\text{exact}}$.

Dirichlet conditions are specified on both the straight and curved boundaries. With the exception of the domain size, the initial mesh is as shown in Figure 5.3—it has 20 elements. We solve on this mesh, then perform uniform h-refinements. We examine best approximation error, DPG error, and convergence rates for a polynomial orders from $k = 1$ to $k = 5$. With the exception of the coarsest meshes, we agree exactly with the best approximation error, and achieve the theoretically optimal convergence rates. The best approximation comparison can be seen in Table 5.3. The convergence rates

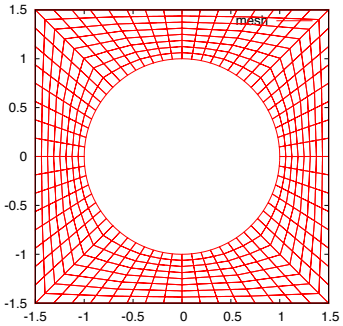


Figure 5.2: Quadratic mesh for the curvilinear Poisson test problem after four refinements (zoom-in near cylinder).

can be seen in Table 5.4. This seems to us to be strong evidence that using curvilinear refinements whose interior edges are straight is a reasonable choice.

k=1						
Ref. number	ϕ		ψ_1		ψ_2	
	actual	best	actual	best	actual	best
0	2.4e-01	2.0e-01	4.8e-01	4.7e-01	4.8e-01	4.7e-01
1	1.4e-01	1.4e-01	1.3e-01	1.3e-01	1.3e-01	1.3e-01
2	3.5e-02	3.5e-02	3.5e-02	3.4e-02	3.5e-02	3.4e-02
3	8.7e-03	8.7e-03	8.7e-03	8.6e-03	8.7e-03	8.6e-03
k=2						
Ref. number	ϕ		ψ_1		ψ_2	
	actual	best	actual	best	actual	best
0	1.9e-01	1.8e-01	1.4e-01	1.4e-01	1.4e-01	1.4e-01
1	1.7e-02	1.7e-02	2.0e-02	2.0e-02	2.0e-02	2.0e-02
2	2.3e-03	2.3e-03	2.5e-03	2.5e-03	2.5e-03	2.5e-03
3	2.9e-04	2.9e-04	3.2e-04	3.2e-04	3.2e-04	3.2e-04
k=3						
Ref. number	ϕ		ψ_1		ψ_2	
	actual	best	actual	best	actual	best
0	9.2e-03	8.8e-03	3.3e-02	3.2e-02	3.3e-02	3.2e-02
1	2.3e-03	2.3e-03	2.2e-03	2.2e-03	2.2e-03	2.2e-03
2	1.4e-04	1.4e-04	1.4e-04	1.4e-04	1.4e-04	1.4e-04
3	8.8e-06	8.8e-06	8.7e-06	8.7e-06	8.7e-06	8.7e-06
k=4						
Ref. number	ϕ		ψ_1		ψ_2	
	actual	best	actual	best	actual	best
0	7.6e-03	7.6e-03	5.5e-03	5.5e-03	5.5e-03	5.5e-03
1	1.7e-04	1.7e-04	1.9e-04	1.9e-04	1.9e-04	1.9e-04
2	5.5e-06	5.5e-06	6.1e-06	6.1e-06	6.1e-06	6.1e-06
3	1.7e-07	1.7e-07	1.9e-07	1.9e-07	1.9e-07	1.9e-07
k=5						
Ref. number	ϕ		ψ_1		ψ_2	
	actual	best	actual	best	actual	best
0	1.9e-04	1.9e-04	8.8e-04	8.6e-04	8.8e-04	8.6e-04
1	1.5e-05	1.5e-05	1.5e-05	1.4e-05	1.5e-05	1.4e-05
2	2.3e-07	2.3e-07	2.2e-07	2.2e-07	2.2e-07	2.2e-07
3	3.5e-09	3.5e-09	3.5e-09	3.5e-09	3.5e-09	3.5e-09

Table 5.3: Poisson manufactured solution on a curvilinear mesh, actual and best approximation errors.

k=1						
Ref. number	ϕ	rate	ψ_1	rate	ψ_2	rate
0	2.4e-01	-	4.8e-01	-	4.8e-01	-
1	1.4e-01	0.80	1.3e-01	1.83	1.3e-01	1.83
2	3.5e-02	2.00	3.5e-02	1.96	3.5e-02	1.96
3	8.7e-03	2.00	8.7e-03	1.99	8.7e-03	1.99
k=2						
Ref. number	ϕ	rate	ψ_1	rate	ψ_2	rate
0	1.9e-01	-	1.4e-01	-	1.4e-01	-
1	1.7e-02	3.43	2.0e-02	2.78	2.0e-02	2.78
2	2.3e-03	2.91	2.5e-03	2.98	2.5e-03	2.98
3	2.9e-04	2.98	3.2e-04	3.00	3.2e-04	3.00
k=3						
Ref. number	ϕ	rate	ψ_1	rate	ψ_2	rate
0	9.2e-03	-	3.3e-02	-	3.3e-02	-
1	2.3e-03	2.03	2.2e-03	3.93	2.2e-03	3.93
2	1.4e-04	4.01	1.4e-04	3.97	1.4e-04	3.97
3	8.8e-06	4.00	8.7e-06	3.99	8.7e-06	3.99
k=4						
Ref. number	ϕ	rate	ψ_1	rate	ψ_2	rate
0	7.6e-03	-	5.5e-03	-	5.5e-03	-
1	1.7e-04	5.50	1.9e-04	4.83	1.9e-04	4.83
2	5.5e-06	4.92	6.1e-06	4.98	6.1e-06	4.98
3	1.7e-07	4.98	1.9e-07	5.00	1.9e-07	5.00
k=5						
Ref. number	ϕ	rate	ψ_1	rate	ψ_2	rate
0	1.9e-04	-	8.8e-04	-	8.8e-04	-
1	1.5e-05	3.70	1.5e-05	5.90	1.5e-05	5.90
2	2.3e-07	6.01	2.2e-07	6.03	2.2e-07	6.03
3	3.5e-09	6.00	3.5e-09	5.99	3.5e-09	5.99

Table 5.4: Poisson: manufactured solution on a curvilinear mesh, L^2 error and h-convergence rates.

Chapter 6

Numerical Experiments for the Stokes Equations

To illustrate the theoretical results proven for the Stokes formulation in Chapter 4, we perform a host of numerical experiments. In Section 6.1, we consider a smooth manufactured solution, first showing optimal convergence when we use the graph norm arising from the analysis, then showing sub-optimal convergence when a naive norm is selected instead. Next, we turn to the classic lid-driven cavity flow problem in Section 6.2, with experiments involving both h- and hp-refinements compared to an overkill solution. In Section 6.3, we solve the backward-facing step problem, again using an overkill mesh to study h- and hp-adaptivity for this problem. Finally, in 6.4, we conclude with consideration of a variation of the graph norm that allows us to achieve a higher convergence rate in the velocity at minimal additional cost.

Recall that the pressure p in the Stokes problem is only determined up to a constant. Following a method described by Bochev and Lehoucq [7], we add a constraint on the pressure that enforces

$$\int_\Omega p = 0,$$

thereby determining the solution uniquely. This constraint is also satisfied by

the manufactured solution used in our experiments.

Before turning to the experiments themselves, we briefly note the expected convergence properties and give a few implementation details. When implementing DPG, we have several choices: what polynomial orders to use for the approximation of fields, traces, and fluxes; how to approximate the optimal test functions, and what norm to use on the test space. We discuss each of these in turn.

Orders of polynomial approximation. As discussed in Chapter 2, once we have fixed a polynomial order k for the field variables on the mesh (or in an element), the natural polynomial order to use is $k+1$ for traces and k for fluxes. As noted previously, we use an enrichment $\Delta k = 2$ for the test space. Note also that we have made no assumptions about the choice of basis functions. The present work uses H^1- and $H(\text{div})$-conforming nodal bases provided by the Intrepid package in Trilinos [56].

Test space norm. The choice of test norm arising from the above analysis is the (adjoint) graph norm:

$$\|(\boldsymbol{\tau}, \boldsymbol{v}, q)\|^2_{\text{graph}} = \|\nabla \cdot \boldsymbol{\tau} - \nabla q\|^2 + \|\nabla \cdot \boldsymbol{v}\|^2 + \|\boldsymbol{\tau} + \nabla \boldsymbol{v}\|^2 + \|\boldsymbol{\tau}\|^2 + \|\boldsymbol{v}\|^2 + \|q\|^2$$

We use this norm in our first experiment, and get the optimal convergence rates for the field variables. In our second experiment, we consider

another choice of test norm, which we refer to as the *naive* test space norm:

$$\|(\boldsymbol{\tau}, \boldsymbol{v}, q)\|^2_{\text{naive}} = \|\boldsymbol{\tau}\|^2 + \|\nabla \cdot \boldsymbol{\tau}\|^2 + \|\boldsymbol{v}\|^2 + \|\nabla \boldsymbol{v}\|^2 + \|q\|^2 + \|\nabla q\|^2 .$$

Note that this is a stronger space than the one generated by the graph norm; that is, if we define

$$V_{\text{graph}} = \{(\boldsymbol{\tau}, \boldsymbol{v}, q) : \|(\boldsymbol{\tau}, \boldsymbol{v}, q)\|_{\text{graph}} < \infty\}, \text{and}$$
$$V_{\text{naive}} = \{(\boldsymbol{\tau}, \boldsymbol{v}, q) : \|(\boldsymbol{\tau}, \boldsymbol{v}, q)\|_{\text{naive}} < \infty\},$$

then $V_{\text{naive}} \subset V_{\text{graph}}$. Specifically, V_{graph} only requires $\nabla \cdot \boldsymbol{\tau} - \nabla q \in \boldsymbol{L}^2$, while V_{naive} requires $\nabla \cdot \boldsymbol{\tau} \in \boldsymbol{L}^2$ *and* $\nabla q \in \boldsymbol{L}^2$.

6.1 Manufactured Solution Experiment

To test the method, we use a manufactured solution following Cockburn et al. [27]

$$u_1 = -e^x(y \cos y + \sin y)$$
$$u_2 = e^x y \sin y$$
$$p = 2\mu e^x \sin y$$

on domain $\Omega = (-1, 1)^2$, taking $\mu = 1$, with uniform quadrilateral meshes of increasing granularity, and examine convergence rates. The L^2 norm of the exact solution for u_1 is 2.53; for u_2, 1.07; for p, 2.81.

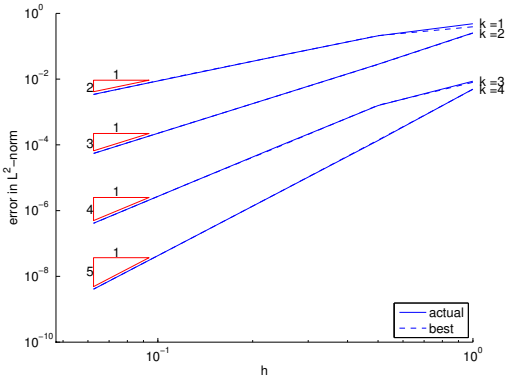
Graph norm experiment. Figures 6.1 and 6.2 show h- and p-convergence results using the graph norm in the test space, for uniform quadrilateral meshes

varying from $k = 1$ to 4 in polynomial order, and from 1×1 to 16×16 elements. The dashed lines in the plots show the error of an L^2 projection of the exact solution (the theoretical best we could achieve)—the lines lie nearly on top of each other. We not only observe optimal convergence rates, but almost exactly achieve the best approximation error.

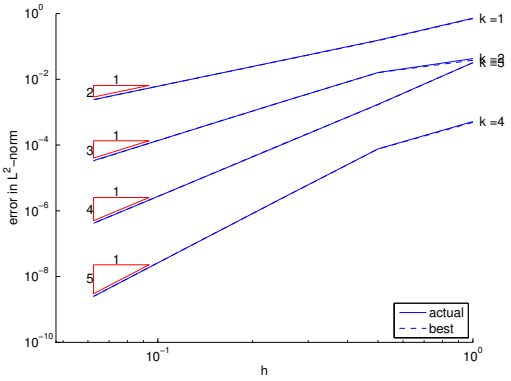
Naive norm experiment. Our second manufactured solution experiment uses the naive norm on the test space. This was the first norm we used when studying DPG formulations of Stokes [74], before we had developed the analysis above, showing why the naive norm might not do as well as the graph norm does.

Figures 6.3 and 6.4 show h- and p-convergence results using the naive norm in the test space, for uniform quadrilateral meshes varying from $k = 1$ to 4 in polynomial order, and from 1×1 to 16×16 elements; we have again plotted for comparison the error in the L^2 projection of the exact solution. As with the graph norm, here we observe optimal convergence rates and almost exactly achieve the best approximation error in velocities u_1 and u_2, but in the pressure p we are sub-optimal by up to two orders of magnitude.

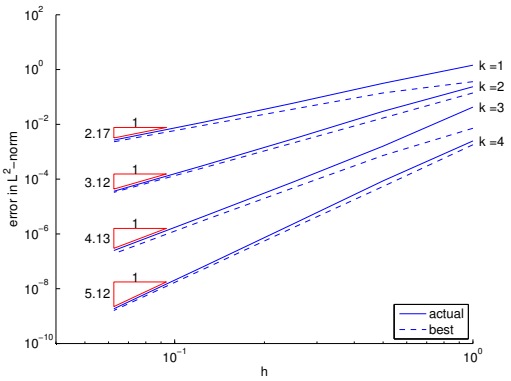
Why do we not see optimal convergence for the naive norm? This is a stronger norm than the graph norm used in our analysis in Chapter 4—consequently, we hypothesize that the corresponding continuous problem is ill-posed, resulting in a mesh-dependent inf-sup constant.



(a) u_1

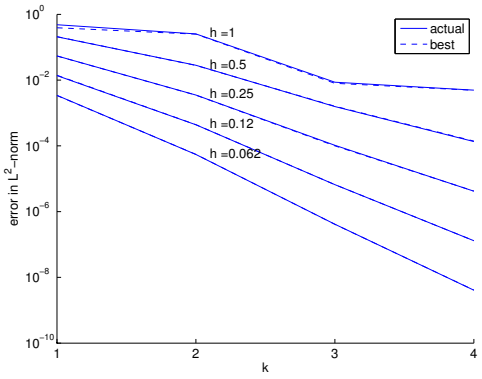


(b) u_2

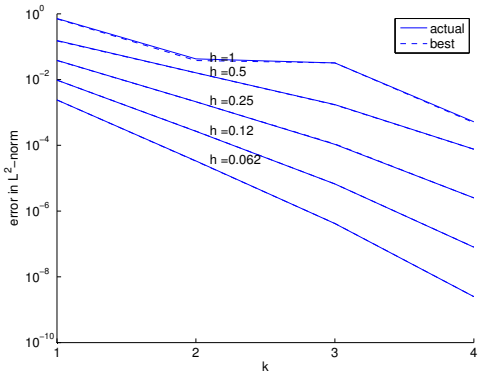


(c) p

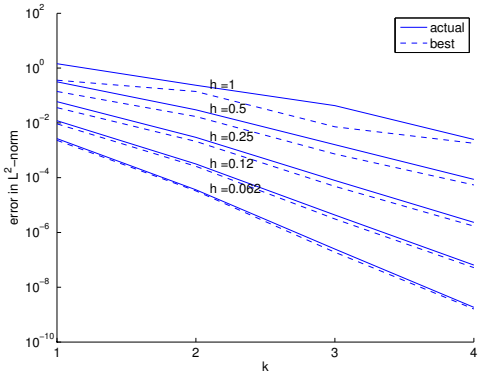
Figure 6.1: h-convergence of u_1, u_2 and p when using the graph norm for the test space. We observe optimal convergence rates, and nearly match the L^2-projection of the exact solution.



(a) u_1

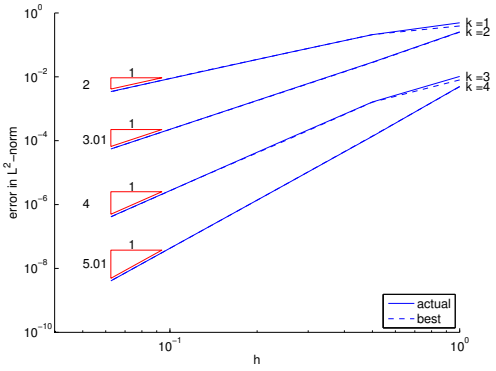


(b) u_2

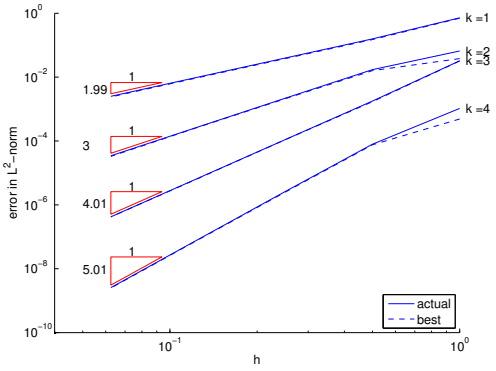


(c) p

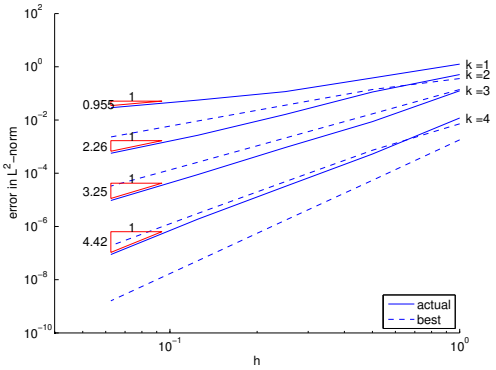
Figure 6.2: p-convergence of u_1, u_2 and p when using the graph norm for the test space. We observe exponential convergence for the finer meshes, and nearly match the L^2-projection of the exact solution.



(a) u_1

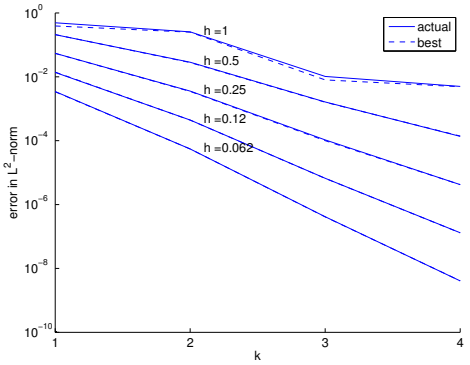


(b) u_2

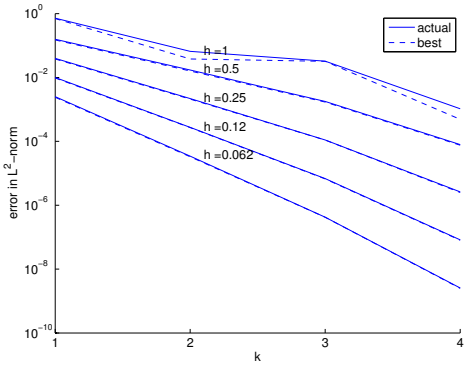


(c) p

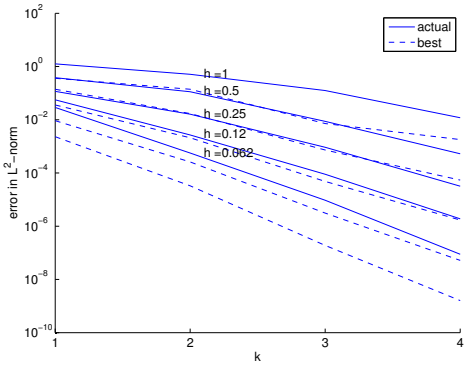
Figure 6.3: h-convergence of u_1, u_2 and p when using the naive norm for the test space. We observe optimal convergence rates (and nearly match the L^2-projection of the exact solution) for u_1 and u_2, but p converges at suboptimal rates.



(a) u_1



(b) u_2



(c) p

Figure 6.4: p-convergence of u_1, u_2 and p when using the naive norm for the test space. We observe exponential convergence for the finer meshes, and nearly match the L^2-projection of the exact solution for u_1 and u_2, but see significantly suboptimal solutions in p.

6.2 Lid-Driven Cavity Flow

A classic test case for Stokes flow is the lid-driven cavity flow problem. Consider a square cavity with an incompressible, viscous fluid, with a lid that moves at a constant rate. The resulting flow will be vorticular; as sketched in Figure 6.5, there will also be so-called *Moffatt eddies* at the corners; in fact, the exact solution will have an infinite number of such eddies, visible at progressively finer scales [66]. Note that the problem as described will have a discontinuity in the fluid velocity at the top corners, and hence its solution will not conform to the spaces we used in our analysis; for this reason, in our experiment we approximate the problem by introducing a thin ramp in the boundary conditions—we have chosen a ramp of width $\frac{1}{64}$. This makes the boundary conditions continuous,[1] so that the solution conforms to the spaces used in the analysis.

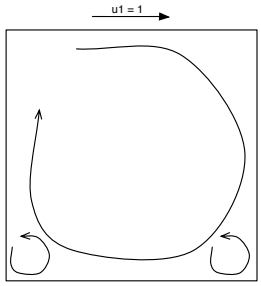


Figure 6.5: Sketch of lid-driven cavity flow.

As described in the introduction, DPG gives us a mechanism for

[1]It is worth noting that these boundary conditions are not exactly representable by many of the coarser meshes used in our experiments. We interpolate the boundary conditions in the discrete space.

measuring the residual error in the dual norm (the very error we seek to minimize) precisely, and we use this to drive adaptivity, by measuring the error $\|e_K\|_V$ for each element K. Both the method and our code allow refinements in h or p or in some combination of h and p. However, we do not yet have a general mechanism for deciding *which* refinement to apply (h or p), once we have decided that a given element should be refined. We run two experiments, one with h-adaptivity and one using an ad hoc hp-adaptive strategy, described below.

Although it is not required by the code, we enforce *1-irregularity* throughout—that is, before an element can be refined twice along an edge, its neighbor along that edge must be refined once. In limited comparisons running the same experiments without enforcing 1-irregularity, this did not appear to make much practical difference.

6.2.1 h-refinement strategy

For h-refinements, our strategy is very simple:

1. Loop through the elements, determining the maximum element error $\|e_{K\max}\|_V$.
2. Refine all elements with error at least 20% of the maximum $\|e_{K\max}\|_V$.

Because the exact solution is unknown, we first solve on an overkill mesh and compare our adaptive solution at each step to the overkill solution. In this experiment, we used quadratic field variables ($k = 2$), a test space

enrichment of 1 relative to the H^1 order (that is, $k_{\text{test}} = k + 2 = 4$) for both the adaptive and overkill solutions. The overkill mesh was 256×256 elements, with 5,576,706 dofs.

The initial mesh was a 2×2 square mesh; we ran seven h-adaptive refinements. We stopped after seven steps to ensure that the resulting mesh was nowhere finer than the overkill solution. At each step, we computed the Euclidean (ℓ_2) norm of the L^2 norm of each of the seven field variables. The final adaptive mesh has 124 elements and 11,202 dofs, and combined L^2 error of 4.4×10^{-4} compared with the overkill mesh. We also ran a few uniform refinements and computed the L^2 error for these compared with the overkill mesh, to show the comparative efficiency of the adaptive refinements. The results are plotted in Figure 6.6.

We also post-processed the results to solve for the stream function ϕ, where $\Delta\phi = \nabla \times \boldsymbol{u}$; the contours of ϕ are the streamlines of the flow. Because the DPG solver in Camellia was the most conveniently available solver, we posed this as a DPG problem. This is not perhaps ideal, in that it can allow discontinuities in ϕ, particularly for coarse meshes. Introducing $\boldsymbol{\psi} = \nabla\phi$, the first order system to solve is

$$\nabla \cdot \boldsymbol{\psi} = \nabla \times \boldsymbol{u} = -\sigma_{12} + \sigma_{21}$$
$$\boldsymbol{\psi} - \nabla\phi = 0$$

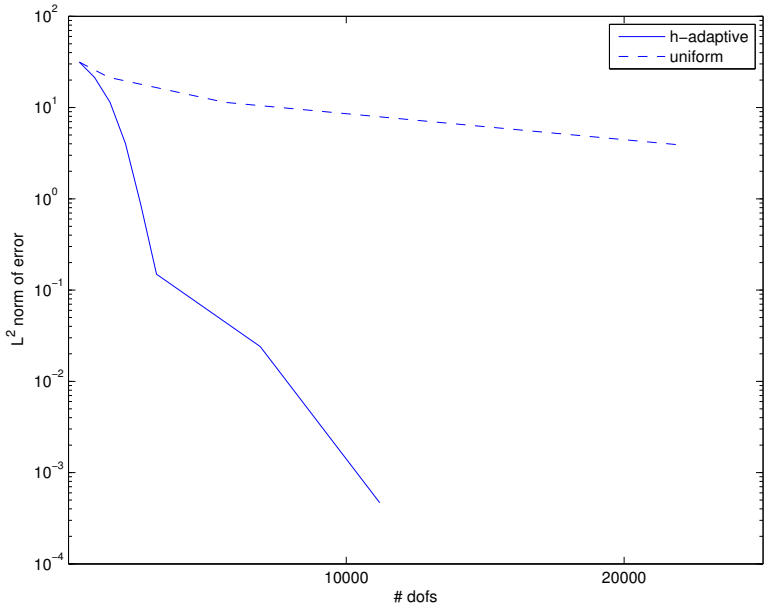


Figure 6.6: Euclidean norm of L^2 error in all field variables in h-adaptive mesh relative to an overkill mesh with 256×256 quadratic elements. The Euclidean norm of all field variables in the exact solution is 6.73.

As usual, we multiply by test functions $(q, \boldsymbol{v})$, integrate by parts, and introduce fluxes and traces on the element boundaries:

$$-(\boldsymbol{\psi}, \nabla q) + \langle \widehat{\psi}_n, q \rangle = (-\sigma_{12} + \sigma_{21}, q)$$
$$(\boldsymbol{\psi}, \boldsymbol{v}) + (\phi, \nabla \cdot \boldsymbol{v}) - \langle \widehat{\phi}, v \rangle = 0$$

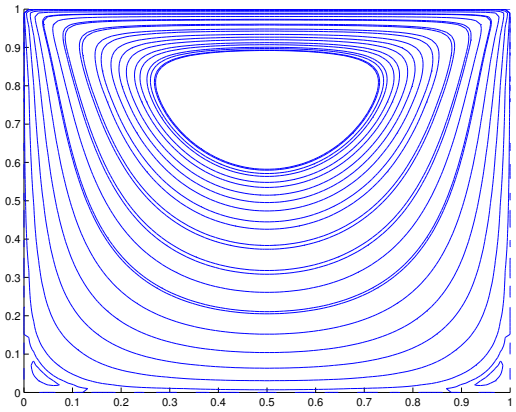
We enforce boundary conditions $\widehat{\phi} = 0$ along the mesh boundaries, as these are solid walls. The streamlines are plotted for the quadratic adaptive mesh described above in Figure 6.7; the first Moffatt eddy can be seen clearly in the zoomed-in plot.

6.2.2 Ad hoc *hp*-refinement strategy

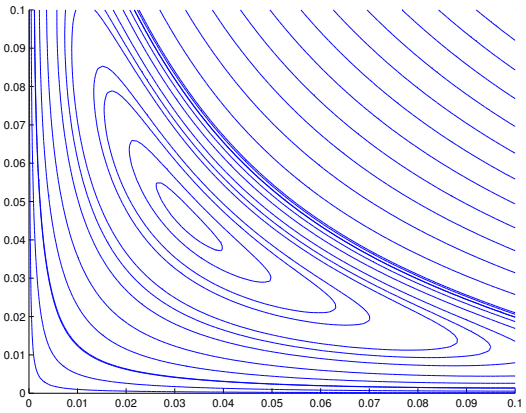
For the *hp* experiment, we adopt a similar strategy; this time, our overkill mesh contains 64×64 quintic elements, and our initial mesh has 2×2 linear elements. We know *a priori* that we should refine in h at the top corners—if only to fully resolve the boundary condition. The strategy is again:

1. Loop through the elements, determining the maximum element error $\|e_{K\max}\|_V$.
2. Refine all elements with error at least 20% of the maximum $\|e_{K\max}\|_V$.

However, this time we must decide whether to refine in h or p. The basic constraints we would like to follow are:



(a) full cavity



(b) lower-left corner

Figure 6.7: Streamlines for the full cavity and for the lower-left corner, on a quadratic mesh after 7 adaptive refinements. The lower-left corner shows the first Moffatt eddy. The final mesh has 124 elements and 11,202 dofs.

- the adaptive mesh must be nowhere finer than the overkill mesh (in h or p), and
- prefer h-refinements at all corners (top and bottom).

So, once the corner elements are as small as the overkill mesh, then they refine in p, and all other elements refine in p until they are quintic, after which they may refine in h.

The primary purpose of this experiment is to demonstrate that the method allows arbitrary meshes of arbitrary, variable polynomial order. The strategy described above clearly depends on *a priori* knowledge of the particular problem we are solving; we have yet to determine a good general strategy for deciding between h- and p-refinements.

We ran 9 refinement steps. The final mesh has 46 elements and 5,986 dofs, compared with 1,223,682 dofs in the overkill mesh. The L^2 error of the adaptive solution compared with the overkill is 8.0×10^{-4}. As in the previous experiment, we also tried running a few uniform h-refinements on the same initial mesh, as a baseline for comparison. The results are plotted in Figure 6.8; the mesh can be seen in Figure 6.9.

6.2.3 Resolving Moffatt Eddies

One way to test and demonstrate the ability of a Stokes solver to resolve fine features of solutions is to examine the resolution of the Moffatt eddies in lid-driven cavity flow. There is an infinite series of these eddies in the lower

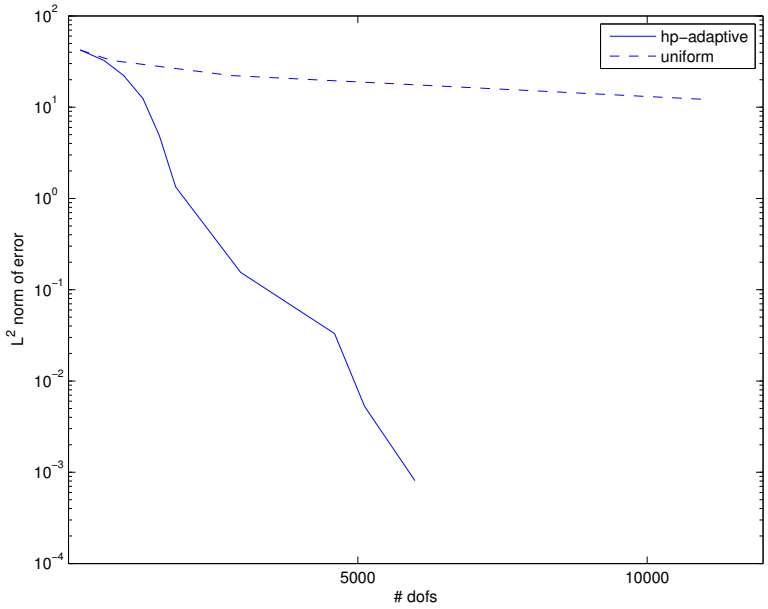


Figure 6.8: Euclidean norm of L^2 error in all field variables in (ad hoc) *hp*-adaptive mesh relative to an overkill mesh with 64×64 quintic elements. The Euclidean norm of all field variables in the exact solution is 6.73; the final mesh has 46 elements and 5,986 dofs.

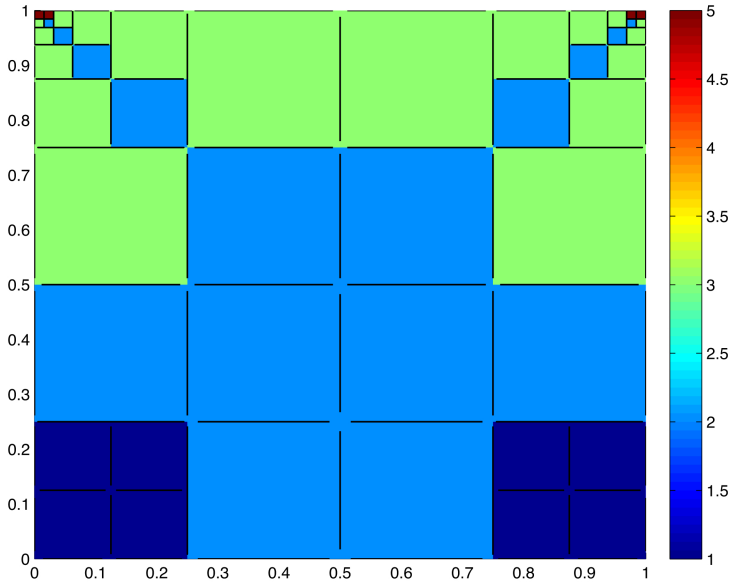


Figure 6.9: Adaptive mesh for ad hoc *hp*-adaptivity strategy after 9 refinement steps. The scale represents the polynomial order of the L^2 variables in the solution. The final mesh has 46 elements and 5,986 dofs.

corners; typical Stokes solvers can resolve the first and perhaps the second; the second is already quite fine, at a scale of approximately 0.006.

Because the Moffatt eddies are weak phenomena when measured with the energy norm, our automatic adaptivity strategy will not prioritize their resolution—instead, it will prioritize the top corners, where the velocity gradients are sharp. Therefore, we adopt a simple ad hoc strategy, based on our standard adaptive strategy, starting with a 2×2 quartic mesh.

1. Mark cells to be refined according to the standard greedy strategy, with threshold of 20%.

2. Reflect cell markings across the horizontal midline—i.e. refine around the lower corners according to the refinement pattern near the upper corners.

It is worth emphasizing that this adaptive strategy is very ad hoc, based on the fortuitous location of the Moffatt eddies and the refinements generated by the standard adaptive strategy. We could easily imagine, however, a goal-oriented adaptive strategy that has as its goal resolution of the vorticity in the lower corners; the present experiment suggests what results we might expect from such a strategy. Instead of the nodal bases provided by Intrepid which we use in most of our experiments, here to improve the conditioning of the local solve we used basis functions derived from the Lobatto shape functions in our test space, and also employed a scaled graph norm, weighted in a way that improves the conditioning of the global solve—more details can be found below

in Section 6.4. We then ran 14 refinements according to the strategy above, resulting in the mesh shown in Figure 6.10. The final mesh had 760 elements and 189,672 dofs, with an h-ratio of 4096 between the largest and smallest elements. The streamline plots for details of the resulting solution are shown

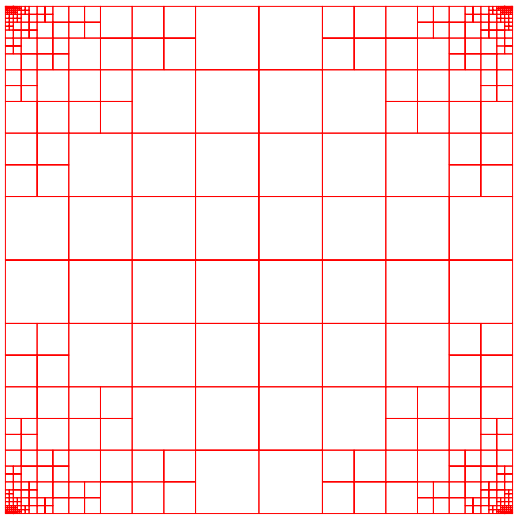
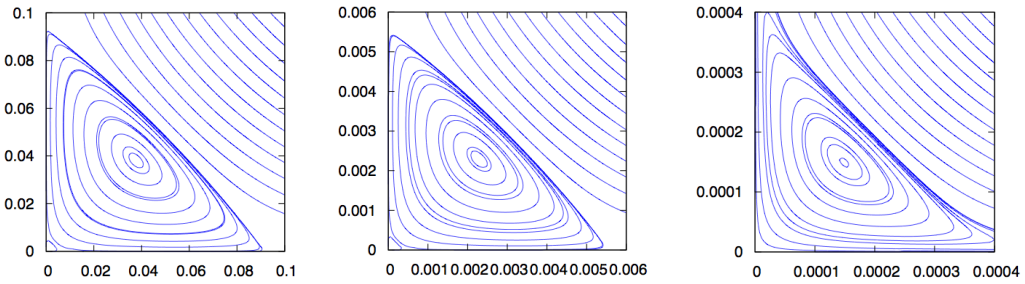


Figure 6.10: Adaptive quartic mesh for ad hoc Moffatt eddy resolution strategy after 14 refinement steps. The mesh has 760 elements and 189,672 dofs, with an h-ratio of 4096 between the largest and smallest elements.

in Figure 6.11; we resolve the first, second, and third Moffatt eddies. These results were computed without static condensation, which we have observed improves conditioning of the global solve. Using static condensation and 16 mesh refinements, we were able to resolve the fourth Moffatt eddy, which is shown in Figure 6.12. The final mesh had 1,192 elements and 296,160 dofs, with an h-ratio of 16384 between the largest and smallest elements.



(a) First Moffatt Eddy (b) Second Moffatt Eddy (c) Third Moffatt Eddy

Figure 6.11: Streamlines for the first, second, and third Moffatt eddies on a quartic mesh after 14 refinement steps.

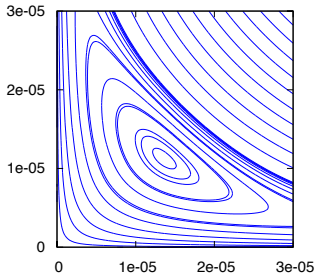


Figure 6.12: Streamlines for the fourth Moffatt eddy on a quartic mesh after 16 refinement steps.

6.2.4 Validation of the Use of the Ramp Boundary Condition

The classical lid-driven cavity problem's solution does not lie in H^1, and the problem we posed, using a "ramp" boundary condition in the velocity to obtain a solution that does lie in H^1 therefore differs from the classical problem. Is this justified? We believe it is, for two reasons. First, in the physical systems modeled by the problem, there will not be point singularities in the velocity—there will be some smooth transition from the zero velocity at the vertical wall to the unit velocity at the lid. Although this smooth transition is unlikely to be captured in any precise way by our ramp boundary condition, it does seem reasonable to suppose that the problem we solve models the physical systems at least as well as the classical model problem.

A second reason for believing that the ramp boundary condition is reasonable is that our solutions appear nearly identical to those found in the literature. For example, one of the reference values reported in the benchmark pseudo-spectral results of Botella and Peyret [9] is the vorticity for the Stokes cavity problem at the point (0,0.95), which they give as 27.27901—note that the point (0,0.95) lies just underneath the right side of the lid, a location close to a region where our boundary conditions differ. Evans et al. [45] also report their computation of this value using divergence-conforming B-splines. Note also that we can use DPG to solve the problem on a fixed mesh without the ramp—the negative effects of the corner singularities are felt primarily in the context of adaptivity.

To verify that the introduction of the ramp in the boundary conditions

has a small effect, we computed the vorticity at $(0, 0.95)$ for several uniform meshes using DPG with the ramp and DPG without the ramp. Table 6.1 lists these alongside values reported by Evans et al. As can be seen in the table, both DPG solutions return values close to each other, and quite close to the benchmark value.

k	h	DPG	DPG (no ramp)	Div-conf. B-splines [45]
1	1/64	19.01363	18.45568	19.04468
1	1/128	23.36717	22.94259	23.29179
1	1/256	25.38294	25.16889	25.32238
2	1/64	27.39212	26.99080	32.81972
2	1/128	27.35573	27.17714	26.48645
3	1/64	27.39660	27.19651	29.92944

Table 6.1: Vorticity computed at (0,0.95) for the Stokes cavity flow problem using DPG with and without the "ramp" in the boundary conditions, and values reported by Evans et al. [45] using divergence-conforming B-splines. The benchmark vorticity value given by Botella and Peyret [9] is 27.27901.

6.3 Backward-facing Step

Another common model problem for Stokes and Navier-Stokes flow involves flow over a backward-facing step, a schematic of which can be seen in Figure 6.13. The corner induces a singularity in the stress, making this a challenging flow to resolve, and one well-suited to adaptivity. One question that arose in early investigations of these flows had to do with the number of vortices behind the step—i.e. whether, as in cavity flow, there is a sequence of progressively smaller eddies in the corner; early numerical studies showed only

one such vortex [6]. Alleborn et. al. [1] performed careful analysis showing that for low Reynolds number flows a series of such eddies does indeed exist.

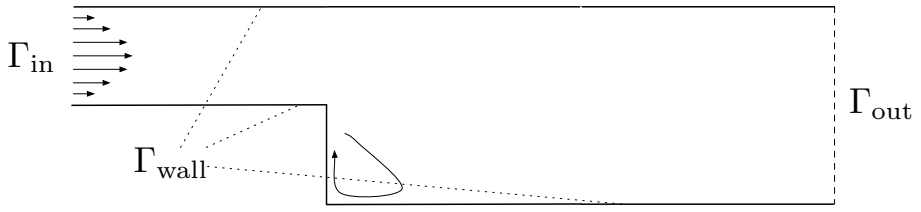


Figure 6.13: Schematic of the backward-facing step problem with a parabolic inflow profile.

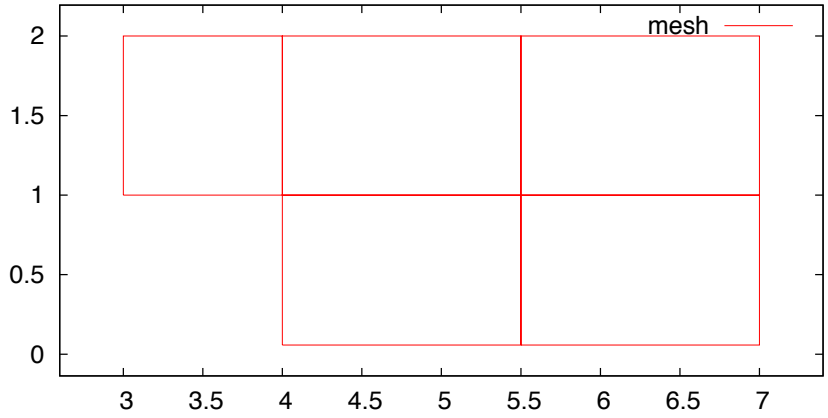


Figure 6.14: Initial mesh for the backward-facing step problem.

We follow Biswas et al. in selecting parameters for the problem [6], principally for conformity with Navier-Stokes experiments that will follow in Chapter 8. We employ an expansion ratio of 1.9423, and use a parabolic inflow

profile with maximal velocity $u_1 = \frac{3}{2}$. We impose inflow conditions $\widehat{\boldsymbol{u}} = \boldsymbol{u}_{\text{in}}$ on the boundary Γ_{in}, no-flow conditions $\widehat{\boldsymbol{u}} = \mathbf{0}$ on Γ_{wall} and "do-nothing" conditions $\widehat{\boldsymbol{t}}_n = \mathbf{0}$ on the outflow Γ_{out}. Note that a consequence of the outflow condition is that the pressure p is uniquely defined—there is no need here for the zero-mean constraint which we required when we set boundary conditions on the velocity alone.

For each of our experiments, we begin with a mesh of five elements, as shown in Figure 6.14. For our first experiment, we took a quartic initial mesh, and refined in h. Just as with lid-driven cavity flow, no analytic solution is available, so we again make use of an overkill solution to measure the error. All error measures reported are scaled relative to the L^2 norm of the overkill solution.

For our h-adaptive experiment, we construct the overkill mesh by beginning with a linear initial mesh as shown in Figure 6.14, and h-refining uniformly 6 times, resulting in a mesh of 49,152 elements of width and 4,185,602 degrees of freedom. We use the same initial mesh for our adaptive mesh, and perform 6 adaptive refinements. At each refinement, we compute the L^2 error of the field variables relative to the overkill solution. We also L^2-project the overkill solution onto the refined mesh, and thus measure the best approximation error. The final adaptive mesh has 5942 degrees of freedom; the results are shown in Figure 6.15. There, it can be seen that the DPG solution is 1 or 2 orders of magnitude off from the best solution on the mesh; however, it is worth noting that the two plots are essentially parallel, suggesting that

in any case DPG is converging to the solution at the best possible rate.

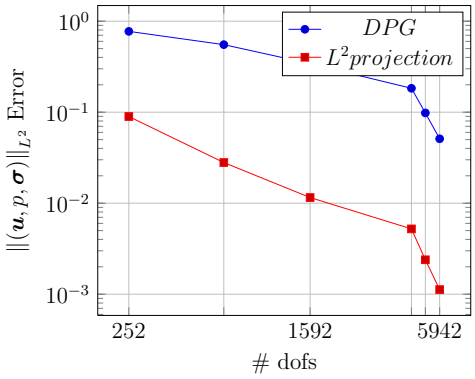


Figure 6.15: Stokes backward-facing step problem: linear h-adaptive solution compared with overkill, and the error of the L^2-projected overkill solution onto the adaptive mesh.

We similarly consider an ad-hoc hp-refinement strategy, now using a quintic overkill solution, uniformly refined 5 times to give us a mesh with 12,288 elements and 3,669,506 degrees of freedom. In the adaptive mesh, we begin with linear elements; when an element is marked for refinement, we refine in p unless the element adjoins either the step corner or the recirculating corner. As with our ad-hoc hp strategy for lid-driven cavity flow, these rules for the choice between h and p are subject to the restriction that the adaptive solution should be nowhere finer than the overkill mesh. We perform 10 refinements. The results are plotted in Figure 6.16. As in the h-refinement experiment, here there is again a considerable gap between the best approximation and the DPG solution; however, here too the rate of convergence appears to be

optimal, and after the sixth refinement there is a considerable acceleration in convergence after the adaptive strategy has resolved the singularity at the step.

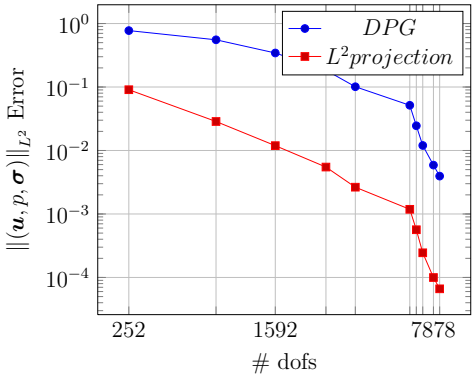


Figure 6.16: Stokes backward-facing step problem: *hp*-adaptive solution compared with overkill, and the error of the L^2-projected overkill solution onto the adaptive mesh.

Why in these cases do we not observe L^2 optimality as we did in the case of our manufactured solution? We believe it is due to the weight on the L^2 terms in the graph norm—in every experiment here, we use unit weights for these. Chan et al. have demonstrated [25] that the graph norm delivers L^2 optimality in the limit as the weights for the L^2 terms go to zero. Moreover, the largest weight that delivers L^2 optimality will in general depend on essentially all the details of the discrete problem: the polynomial orders of test and trial spaces, the geometry, and the boundary conditions—so it is unsurprising that the Stokes graph norm with unit weight would work optimally for one problem,

and only converge at the optimal *rate* for another. (It is the optimal rate that is guaranteed by the theory.) We hope to test our hypothesis by experimenting with smaller weights on the L^2 terms in the future.

6.4 Velocity "Super-convergence": a Scaled Graph Norm

We conclude this chapter with a somewhat speculative idea; we include it chiefly because of the results it has given us—among these, the ability to resolve the third and fourth Moffatt eddies in the lid-driven cavity problem above. We have assumed non-dimensionalization of the Stokes equations with which we began, and this has been the implicit justification for combinations of terms such as the L^2 norm of all variables,

$$\|(\boldsymbol{u}, p, \boldsymbol{\sigma})\|_U^2 := \|\boldsymbol{u}\|^2 + \|\boldsymbol{\sigma}\|^2 + \|p\|^2 , \tag{6.4.1}$$

which is the norm in which we sought optimality by using the graph norm as our test space norm. If we were instead considering dimensional equations, then such combinations of terms would be physically unreasonable, and we would need to add some scaling terms—for example, we might use

$$\|(\boldsymbol{u}, p, \boldsymbol{\sigma})\|_U^2 := \frac{1}{L^2}\|\boldsymbol{u}\|^2 + \|\boldsymbol{\sigma}\|^2 + \|p\|^2 ,$$

where L is some length scale. If we now consider the domain to be an individual element, then a length scale $L = h$ may be appropriate, giving us

$$\|(\boldsymbol{u}, p, \boldsymbol{\sigma})\|_U^2 := \frac{1}{h^2}\|\boldsymbol{u}\|^2 + \|\boldsymbol{\sigma}\|^2 + \|p\|^2 . \tag{6.4.2}$$

If we follow the same reasoning as we did for the graph norm, we have

$$\|(\boldsymbol{v}, q, \boldsymbol{\tau})\|_V^2 = \|\nabla \cdot \boldsymbol{v}\|^2 + \|\boldsymbol{\tau} - \nabla \boldsymbol{v}\|^2 + \|h\nabla q + h\nabla \cdot \boldsymbol{\tau}\|^2 + [\ L^2 \text{ terms}]$$

To determine appropriate weights for the L^2 terms, let us think of the test variables in an adjoint sense—that is, $\boldsymbol{v}$ is a velocity, q a pressure, and $\boldsymbol{\tau}$ a velocity gradient. If we then seek a weighting of these terms that respects the units in the adjoint (which we are now thinking of as dimensional), then the natural L^2 terms are

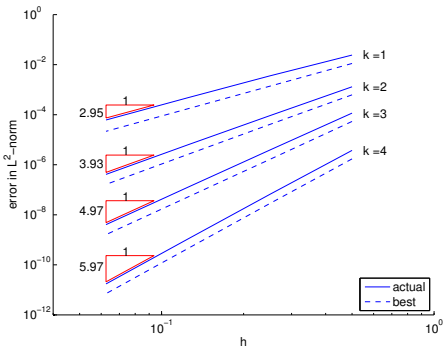
$$\left\| \frac{1}{h} \boldsymbol{v} \right\|^2 + \|q\|^2 + \|\boldsymbol{\tau}\|^2 .$$

A few observations are in order. First of all, both the norm in which we seek optimality and the test norm are now mesh-dependent. Second, the $\frac{1}{h^2}$ weight in front of the $\boldsymbol{u}$ term affects our expected convergence rates. Recall that we chose our polynomial orders so that we would converge at uniform rates in $\|\cdot\|_U$; when this was given by Equation (6.4.1), polynomial order k was appropriate for all the field variables—this allowed an expected convergence rate of h^{k+1} in all variables. Now that $\|\cdot\|_U$ is given by Equation (6.4.2), we require a polynomial order of $k+1$ for u to achieve the same result. Supposing that $\|\cdot\|_U$ does in fact converge at a rate of h^{k+1}, observe that this implies that $\frac{1}{h}\|\boldsymbol{u}\|$ must converge as h^{k+1}. If so, then $\|\boldsymbol{u}\|$ converges at rate h^{k+2}, so that the extra cost of a higher-order velocity gains us a higher rate of convergence. Moreover, since the only change is to the field variables, this extra cost is minimal: when we use static condensation, the global system will

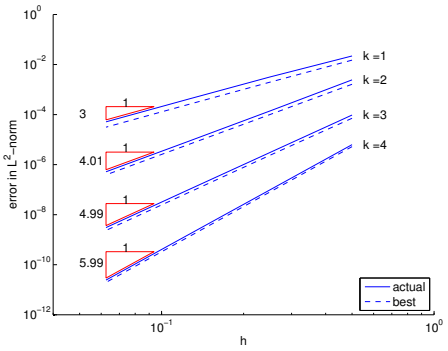
remain exactly the same size, and the only additional cost is in the local static condensation step.

We run one simple numerical experiment with the scaled graph norm: we again solve the manufactured solution given in Section 6.1, now using order $k+1$ polynomials for the velocity field variables. The results are plotted in Figure 6.17; we achieve the predicted convergence rates.

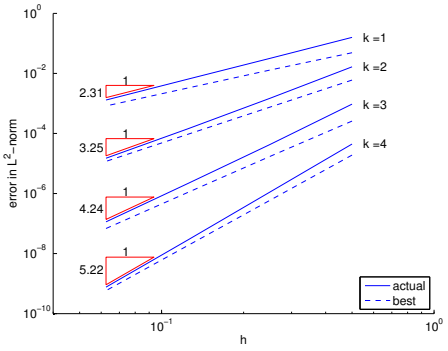
We will revisit the scaled graph norm in Chapter 9; there, we will perform some experiments that suggest this norm strikes a balance between the conditioning of the local solves and the conditioning of the global solves.



(a) u_1



(b) u_2



(c) p

Figure 6.17: h-convergence for smooth manufactured solution (Cockburn et al. [27]) with a scaled graph norm: u_1, u_2, and p. Dashed lines: best approximation error. Note: velocity best approximations are taken in the enriched velocity space—i.e. these converge at rate $k + 2$.

Chapter 7

The Oseen Equations

Whereas the Navier-Stokes equations differ from the Stokes equations in the addition of a convective term $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$, the Oseen equations employ a linear approximation of this term, namely $\boldsymbol{U} \cdot \nabla \boldsymbol{u}$, where the background flow $\boldsymbol{U}$ is taken to be divergence-free. If the exact solution to the Navier-Stokes equations is known, one may take $\boldsymbol{U} = \boldsymbol{u}$; in this case, the stability of an Oseen solver can be understood as the stability of an equivalent linearized Navier-Stokes solver in the immediate vicinity of the exact solution. This is precisely the study we engage in in the present chapter.

7.1 Oseen Equations

Consider the strong form of the Oseen equations:

$$-\nabla p + \mu \Delta \boldsymbol{u} = \boldsymbol{f} + \boldsymbol{U} \cdot \nabla \boldsymbol{u}$$

$$\nabla \cdot \boldsymbol{u} = 0,$$

where the background flow velocity $\boldsymbol{U}$ is taken to be divergence-free. Noting that $\Delta\boldsymbol{u} = \nabla\cdot(\nabla\boldsymbol{u})$, introduce $\boldsymbol{\sigma} = \mu\nabla\boldsymbol{u}$:

$$\begin{aligned} -\nabla p + \nabla\cdot\boldsymbol{\sigma} - \boldsymbol{U}\cdot\nabla\boldsymbol{u} &= \boldsymbol{f} \\ \frac{1}{\mu}\boldsymbol{\sigma} - \nabla\boldsymbol{u} &= 0 \\ \nabla\cdot\boldsymbol{u} &= 0 \end{aligned}$$

Multiplying by test functions $(\boldsymbol{v}, \boldsymbol{\tau}, q)$, we have:

$$\begin{aligned} (-\nabla p, \boldsymbol{v}) + (\nabla\cdot\boldsymbol{\sigma}, \boldsymbol{v}) - ((\boldsymbol{U}\cdot\nabla)\boldsymbol{u}, \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v}) \\ \left(\frac{1}{\mu}\boldsymbol{\sigma}, \boldsymbol{\tau}\right) - (\nabla\boldsymbol{u}, \boldsymbol{\tau}) &= 0 \\ (\nabla\cdot\boldsymbol{u}, q) &= 0. \end{aligned}$$

Integrating the Stokes terms by parts, we have

$$\begin{aligned} \langle -p, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\Gamma_h} + (p, \nabla\cdot\boldsymbol{v})_{\Omega_h} + \langle\boldsymbol{\sigma}\boldsymbol{n}, \mu\boldsymbol{v}\rangle_{\Gamma_h} - (\boldsymbol{\sigma}, \nabla(\mu\boldsymbol{v}))_{\Omega_h} & \\ -((\boldsymbol{U}\cdot\nabla)\boldsymbol{u}, \boldsymbol{v}) &= (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h} \\ (\boldsymbol{\sigma}, \boldsymbol{\tau})_{\Omega_h} - \langle\boldsymbol{u}, \boldsymbol{\tau}\boldsymbol{n}\rangle_{\Gamma_h} + (\boldsymbol{u}, \nabla\cdot\boldsymbol{\tau})_{\Omega_h} &= 0 \\ \langle\boldsymbol{u}\cdot\boldsymbol{n}, q\rangle_{\Gamma_h} - (\boldsymbol{u}, \nabla q)_{\Omega_h} &= 0. \end{aligned}$$

Now, we have a choice in how we remove the trial-space derivative on the convective term. We may either use the definition $\boldsymbol{\sigma} = \mu\nabla\boldsymbol{u}$ or we may integrate the new term by parts. Since the latter choice is more general (not all our Stokes formulations include a field variable corresponding to the velocity gradient), and since this is perhaps more in keeping with the spirit of the

ultra-weak formulation, this is what we choose to do. We have:

$$\begin{aligned}((\boldsymbol{U}\cdot\nabla)\boldsymbol{u},\boldsymbol{v}) &= (U_j u_{i,j}, v_i) = (u_{i,j}, v_i U_j) = (\nabla\boldsymbol{u}, \boldsymbol{v}\otimes\boldsymbol{U}) \\ &= -(u_i, (v_i U_j)_{,j}) + \langle u_i, U_j v_i n_j\rangle \\ &= -(u_i, v_{i,j}U_j + v_i U_{j,j}) + \langle u_i, U_j v_i n_j\rangle \\ &= -(\boldsymbol{u}, (\boldsymbol{U}\cdot\nabla)\boldsymbol{v} + \boldsymbol{v}(\nabla\cdot\boldsymbol{U})) + \langle \boldsymbol{u}, (\boldsymbol{U}\cdot\boldsymbol{n})\,\boldsymbol{v}\rangle \\ &= -(\boldsymbol{u}, (\boldsymbol{U}\cdot\nabla)\boldsymbol{v}) + \langle \boldsymbol{u}, (\boldsymbol{U}\cdot\boldsymbol{n})\,\boldsymbol{v}\rangle,\end{aligned}$$

where the final equality comes from the divergence-free condition on $\boldsymbol{U}$. Replacing the field variables on the skeleton by new trace unknowns and defining traction $\boldsymbol{t_n} = (\boldsymbol{\sigma} - p\boldsymbol{I})\boldsymbol{n}$, we have

$$\begin{aligned}\langle\widehat{\boldsymbol{t}}_{\boldsymbol{n}}, \boldsymbol{v}\rangle_{\Gamma_h} + (p, \nabla\cdot\boldsymbol{v})_{\Omega_h} - (\boldsymbol{\sigma}, \nabla\boldsymbol{v})_{\Omega_h} + (\boldsymbol{u}, (\boldsymbol{U}\cdot\nabla)\boldsymbol{v}) - \langle\widehat{\boldsymbol{u}}, (\boldsymbol{U}\cdot\boldsymbol{n})\,\boldsymbol{v}\rangle &= (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h} \\ \left(\frac{1}{\mu}\boldsymbol{\sigma}, \boldsymbol{\tau}\right)_{\Omega_h} - \langle\widehat{\boldsymbol{u}}, \boldsymbol{\tau}\boldsymbol{n}\rangle_{\Gamma_h} + (\boldsymbol{u}, \nabla\cdot\boldsymbol{\tau})_{\Omega_h} &= 0 \\ \langle\widehat{\boldsymbol{u}}\cdot\boldsymbol{n}, q\rangle_{\Gamma_h} - (\boldsymbol{u}, \nabla q)_{\Omega_h} &= 0.\end{aligned}$$

7.2 Test Norm

We use the graph norm, which seeks optimality in the L^2 norm of the field variables,

$$\|(\boldsymbol{u}, p, \boldsymbol{\sigma})\|_U^2 := \|\boldsymbol{u}\|^2 + \frac{1}{\mu^2}\|\boldsymbol{\sigma}\|^2 + \|p\|^2. \tag{7.2.1}$$

For our Oseen formulation, the graph norm is given by is given by

$$\begin{aligned}\|(\boldsymbol{v}, q, \boldsymbol{\tau})\|_V^2 := \|\nabla\cdot\boldsymbol{v}\|^2 + \|\boldsymbol{\tau} - \mu\nabla\boldsymbol{v}\|^2 + \|\nabla q + \nabla\cdot\boldsymbol{\tau} + (\boldsymbol{U}\cdot\nabla)\boldsymbol{v}\|^2 \\ + \|\boldsymbol{v}\|^2 + \|q\|^2 + \|\boldsymbol{\tau}\|^2.\end{aligned}$$

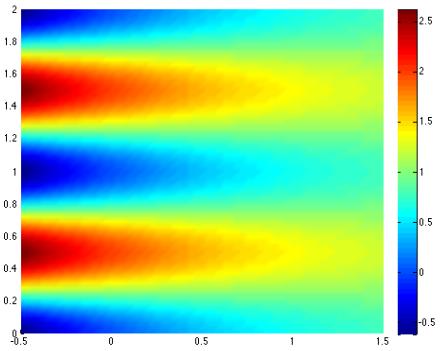
7.3 Numerical Results: Kovasznay Flow

A common test case for Navier-Stokes is an analytic solution due to Kovasznay [60]:

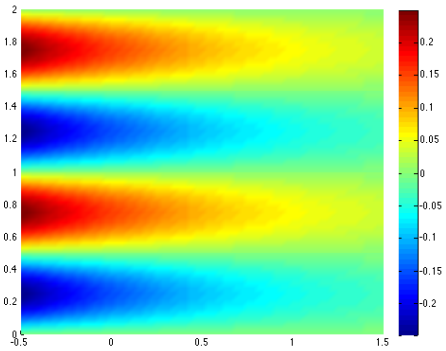
$$\begin{aligned} u_1 &= 1 - e^{\lambda x}\cos(2\pi y) \\ u_2 &= \frac{\lambda}{2\pi} e^{\lambda x}\sin(2\pi y) \\ p &= \frac{1}{2}e^{2\lambda x} + C \end{aligned}$$

where $\lambda = \frac{\mathrm{Re}}{2} - \sqrt{\left(\frac{\mathrm{Re}}{2}\right)^2 + (2\pi)^2}$. We use $\Omega = (-0.5, 1.5)\times(0,2)$ as our domain, and choose the constant C so that p has zero average on Ω.

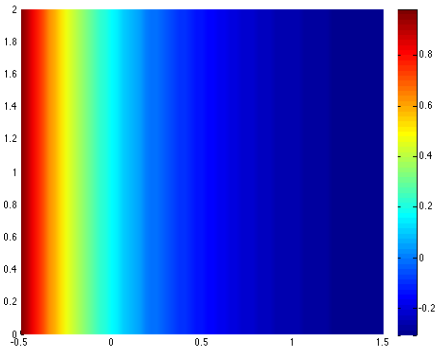
A standard test case for Kovasznay flow is $\mathrm{Re} = 40$; the solution for this case is plotted in Figure 7.1. For our first test, we use the standard graph norm and $k = 1$ to 4 and $\Delta k = 2$, on meshes from 1×1 to 64×64 elements. Figure 7.2 shows the velocity error in the DPG solution as well as the L^2-projection exact solution (the best approximation); the results are essentially perfect. The pressure results are shown in Figure 7.3—here, there is a gap between the best error and that achieved by DPG, but the two are generally close, and DPG converges at appropriate rates. Finally, the euclidean combination of all field variables is shown in Figure 7.4—since this is the norm we sought optimality in by using the graph norm, this is the "fairest" one to examine, and here the results are again essentially perfect. The explanation, then, for the sub-optimal pressure results is simply that the error in other components dominates.



(a) u_1



(b) u_2



(c) p

Figure 7.1: Kovasznay flow for $\mathrm{Re} = 40$: u_1, u_2 and p.

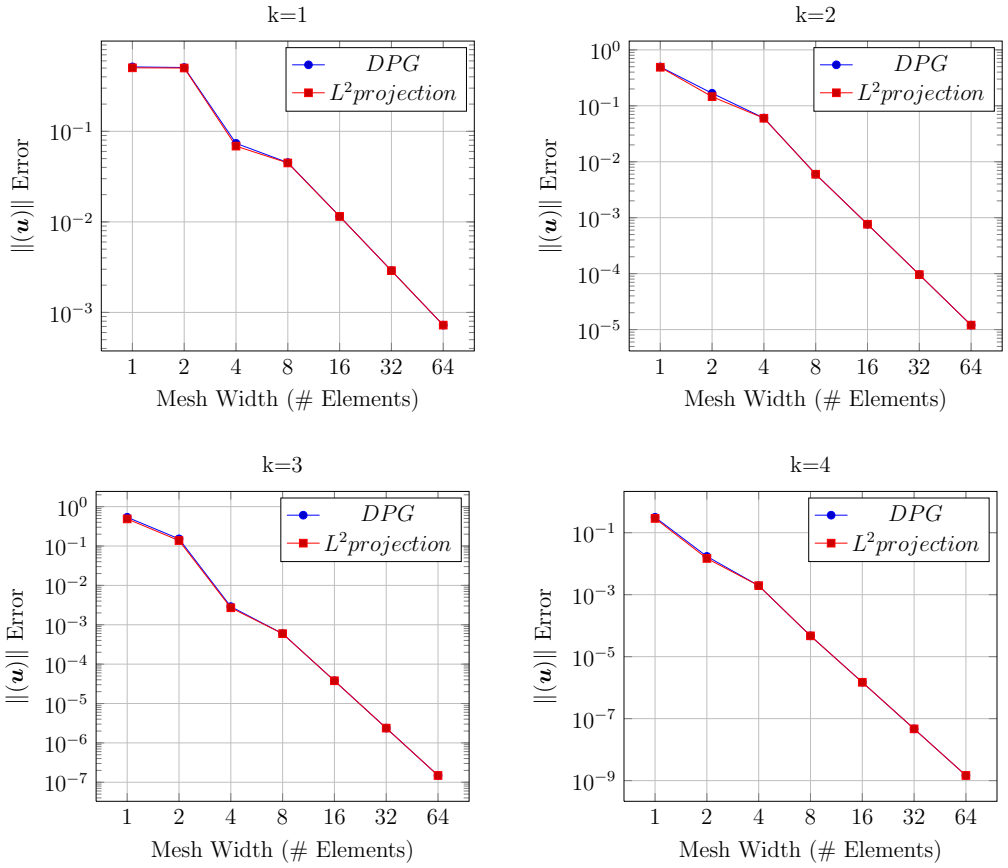


Figure 7.2: Kovasznay flow results for the DPG Oseen formulation using the standard graph norm with Re = 40: velocity error.

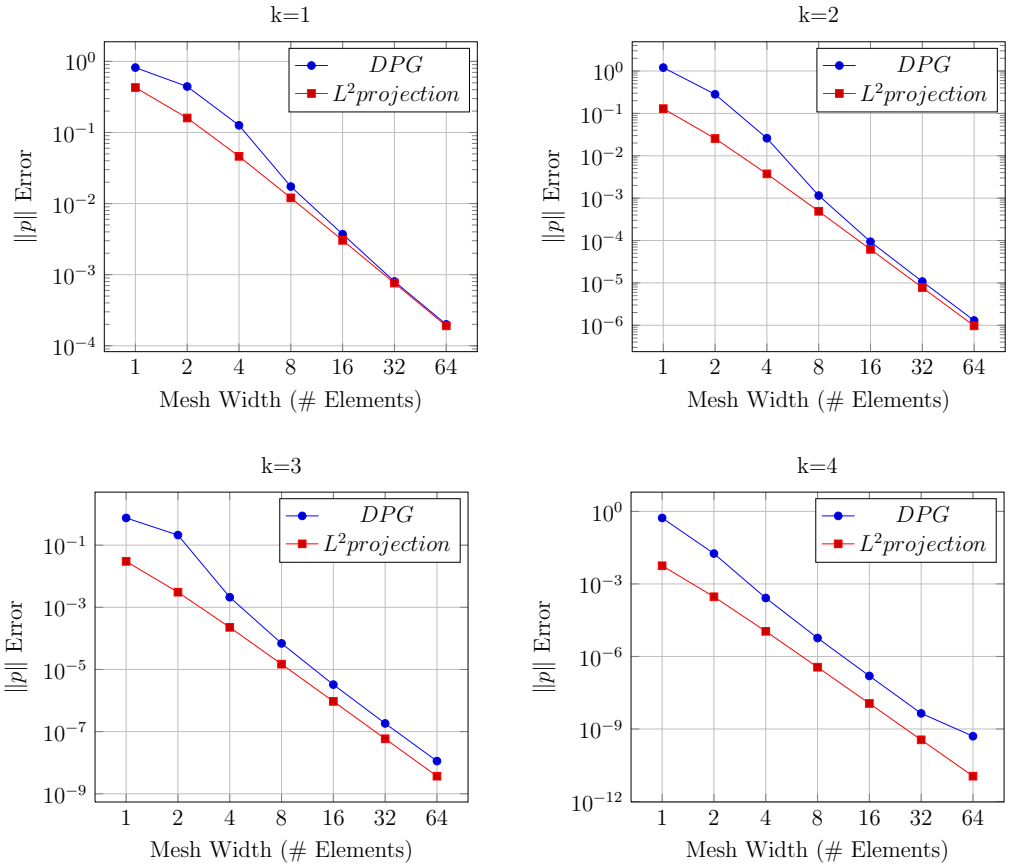


Figure 7.3: Kovasznay flow results for the DPG Oseen formulation using the standard graph norm with Re = 40: pressure error.

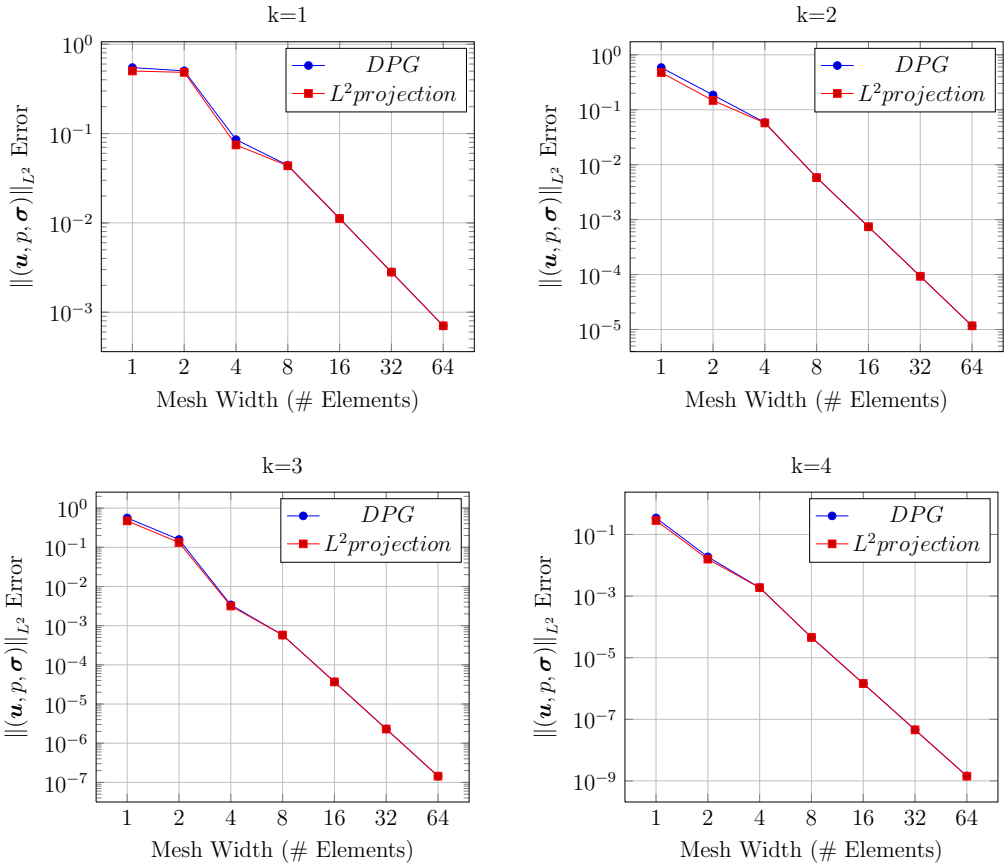


Figure 7.4: Kovasznay flow results for the DPG Oseen formulation using the standard graph norm with Re = 40: L^2 error of all field variables.

Chapter 8

The Navier-Stokes Equations

We now turn to the final set of flow equations considered in this dissertation, the steady state Navier-Stokes equations. We begin by detailing our DPG formulation for the linearized equations in Section 8.1. We then turn to numerical experiments: Kovasznay flow in Section 8.2.1, the lid-driven cavity flow problem in Section 8.2.2, the backward-facing step problem in Section 8.2.3, and finally flow past a cylinder in Section 8.2.4.

8.1 VGP Navier-Stokes Formulation

To derive a Navier-Stokes formulation corresponding to our Stokes VGP formulation, we recall that the Navier-Stokes equations may be written

$$-\nabla p + \mu \nabla \cdot \boldsymbol{\sigma} = \boldsymbol{f} + \boldsymbol{u} \cdot \nabla \boldsymbol{u},$$
$$\boldsymbol{\sigma} - \nabla \boldsymbol{u} = 0,$$
$$\nabla \cdot \boldsymbol{u} = 0.$$

Noting that the convective term may be written $\boldsymbol{u} \cdot \nabla \boldsymbol{u} = \boldsymbol{u} \cdot \boldsymbol{\sigma}$, we immediately see that the corresponding (nonlinear) Navier-Stokes formulation

is

$$\begin{aligned}\langle \widehat{\boldsymbol{t}_{\boldsymbol{n}}}, \boldsymbol{v}\rangle_{\Gamma_h} + (p, \nabla\cdot\boldsymbol{v})_{\Omega_h} + (\boldsymbol{\sigma}, \nabla(\mu\boldsymbol{v}))_{\Omega_h} - (\boldsymbol{u}\cdot\boldsymbol{\sigma}, \boldsymbol{v})_{\Omega_h} &= (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h},\\ (\boldsymbol{\sigma}, \boldsymbol{\tau})_{\Omega_h} - \langle \widehat{\boldsymbol{u}}, \boldsymbol{\tau}\boldsymbol{n}\rangle_{\Gamma_h} + (\boldsymbol{u}, \nabla\cdot\boldsymbol{\tau})_{\Omega_h} &= 0,\\ \langle \widehat{\boldsymbol{u}}\cdot\boldsymbol{n}, q\rangle_{\Gamma_h} - (\boldsymbol{u}, \nabla q)_{\Omega_h} &= 0.\end{aligned}$$

If we define the Stokes bilinear formulation as $b_{\text{Stokes}}(u, v) = l_{\text{Stokes}}(v)$, and linearize about $(\boldsymbol{u} + \Delta\boldsymbol{u}, \boldsymbol{\sigma} + \Delta\boldsymbol{\sigma}, p + \Delta p)$, we then have

$$b_{\text{Stokes}}(\Delta u, v) - (\Delta\boldsymbol{u}\cdot\boldsymbol{\sigma} + \boldsymbol{u}\cdot\Delta\boldsymbol{\sigma}, \boldsymbol{v})_{\Omega_h} = (\boldsymbol{f}, \boldsymbol{v})_{\Omega_h} - b_{\text{Stokes}}(u, v) + (\boldsymbol{u}\cdot\boldsymbol{\sigma}, \boldsymbol{v})_{\Omega_h}.$$

It is worth being precise about what this means in terms of scalar components. $\boldsymbol{\sigma} = \nabla\boldsymbol{u}$, and we define $\binom{\sigma_{11}}{\sigma_{12}} = \nabla u_1$, $\binom{\sigma_{21}}{\sigma_{22}} = \nabla u_2$. Then

$$\begin{aligned}\Delta\boldsymbol{u}\cdot\boldsymbol{\sigma} = \begin{pmatrix}\Delta u_1\\ \Delta u_2\end{pmatrix}\cdot\nabla\boldsymbol{u} &= \Delta u_1\frac{\partial}{\partial x}\boldsymbol{u} + \Delta u_2\frac{\partial}{\partial y}\boldsymbol{u}\\ &= \Delta u_1\begin{pmatrix}\sigma_{11}\\ \sigma_{21}\end{pmatrix} + \Delta u_2\begin{pmatrix}\sigma_{12}\\ \sigma_{22}\end{pmatrix}.\end{aligned}$$

Thus the v_1 equation is augmented with

$$-(\Delta u_1\sigma_{11} + \Delta u_2\sigma_{12} + u_1\Delta\sigma_{11} + u_2\Delta\sigma_{12}, v_1),$$

and the v_2 equation with

$$-(\Delta u_1\sigma_{21} + \Delta u_2\sigma_{22} + u_1\Delta\sigma_{21} + u_2\Delta\sigma_{22}, v_2).$$

The corresponding graph norm is

$$\|(\boldsymbol{v}, \boldsymbol{\tau}, q)\|^2 = \left\|\nabla \cdot \boldsymbol{\tau} - \nabla q + \boldsymbol{\sigma}^T \boldsymbol{v}\right\|^2 + \left\|\nabla(\mu \boldsymbol{v}) + \boldsymbol{\tau} + \boldsymbol{u}\boldsymbol{v}^T\right\|^2 + \|\nabla \cdot \boldsymbol{v}\|^2 + \|\boldsymbol{v}\|^2 + \|\boldsymbol{\tau}\|^2 + \|q\|^2 .$$

Artificial time-stepping. We have found that for certain problems with higher Reynolds numbers—notably the lid-driven cavity flow problem with Reynolds numbers above 1000—artificial time-stepping allows us to converge to a solution when otherwise we would not be able to. Recall that the transient Navier-Stokes equations have a time derivative $\frac{\partial \boldsymbol{u}}{\partial t}$ in the momentum equation. Artificial time-stepping treats the nonlinear step as a discrete time step of size Δt; so that we augment our linearized bilinear form with the term

$$\left(-\frac{\Delta \boldsymbol{u}}{\Delta t}, \boldsymbol{v}\right).$$

Appropriate choices of Δt are determined experimentally. A common practice is to use small values of Δt for the first nonlinear iterates, and gradually increase the time step as the iteration converges. In our present work, we have found that fixed time steps with $\Delta t = 1$ suffice.

8.2 Numerical Experiments

8.2.1 Kovasznay Flow

Recall Kovasznay's analytic solution to the Navier-Stokes equations, which we examined in the context of the Oseen equations in Section 7.3:

$$\begin{aligned} u_1 &= 1 - e^{\lambda x}\cos(2\pi y), \\ u_2 &= \frac{\lambda}{2\pi} e^{\lambda x}\sin(2\pi y), \\ p &= \frac{1}{2}e^{2\lambda x} + C, \end{aligned}$$

where $\lambda = \frac{\mathrm{Re}}{2} - \sqrt{\left(\frac{\mathrm{Re}}{2}\right)^2 + (2\pi)^2}$.

As with Oseen, we perform convergence studies for $\mathrm{Re} = 40$, now for $k = 1$ to 4, with mesh sizes ranging from 1×1 to 32×32. We use a zero initial guess, and the convergence criterion that the L^2 norm of the field variables in the Newton-Raphson step should be less than 10^{-12}. Perhaps unsurprisingly, the results are basically identical with those we had for the equivalent Oseen experiment. Figure 8.1 shows the velocity error in the DPG solution as well as the L^2-projection exact solution (the best approximation); the results are essentially perfect. The pressure results are shown in Figure 8.2—here again, there is a gap between the best error and that achieved by DPG, but the two are generally close, and DPG converges at appropriate rates. Finally, the euclidean combination of all field variables is shown in Figure 8.3—as in the Oseen experiment, here the results are once again essentially perfect.

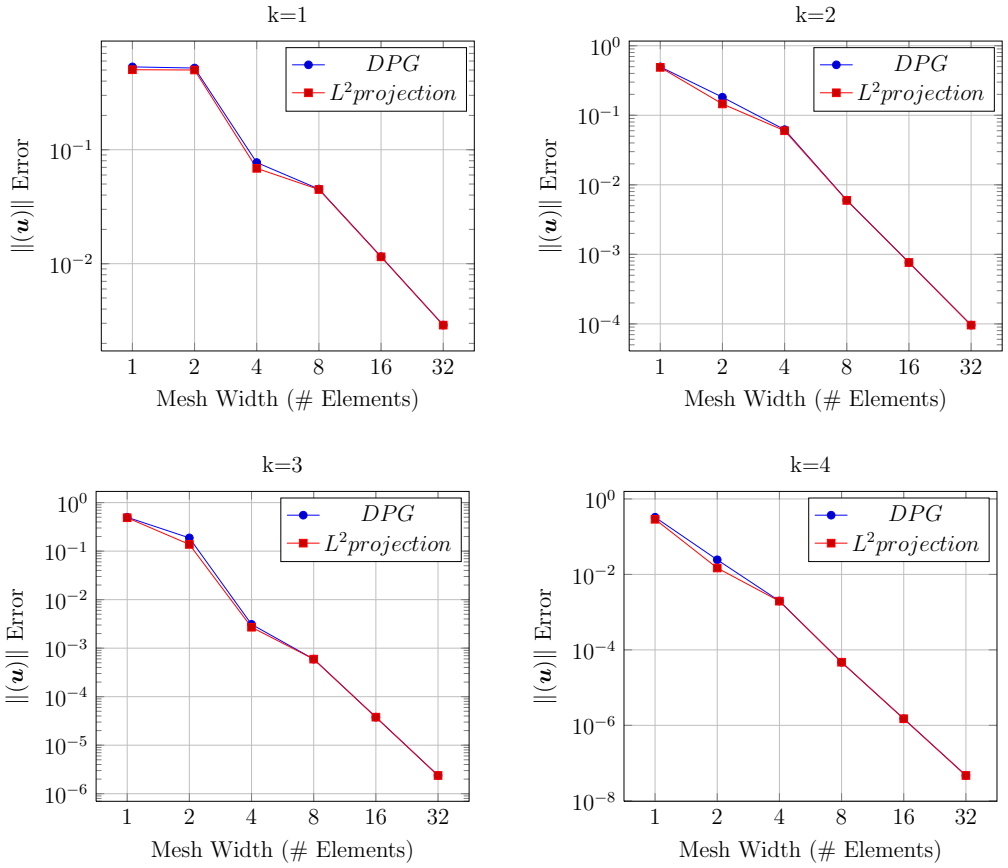


Figure 8.1: Kovasznay flow results for the DPG Navier-Stokes formulation using the standard graph norm with Re = 40: velocity error.

8.2.2 Lid-Driven Cavity Flow

Recall the lid-driven cavity problem which we solved in the context of the Stokes equations in Section 6.2, a schematic of which is shown in Figure

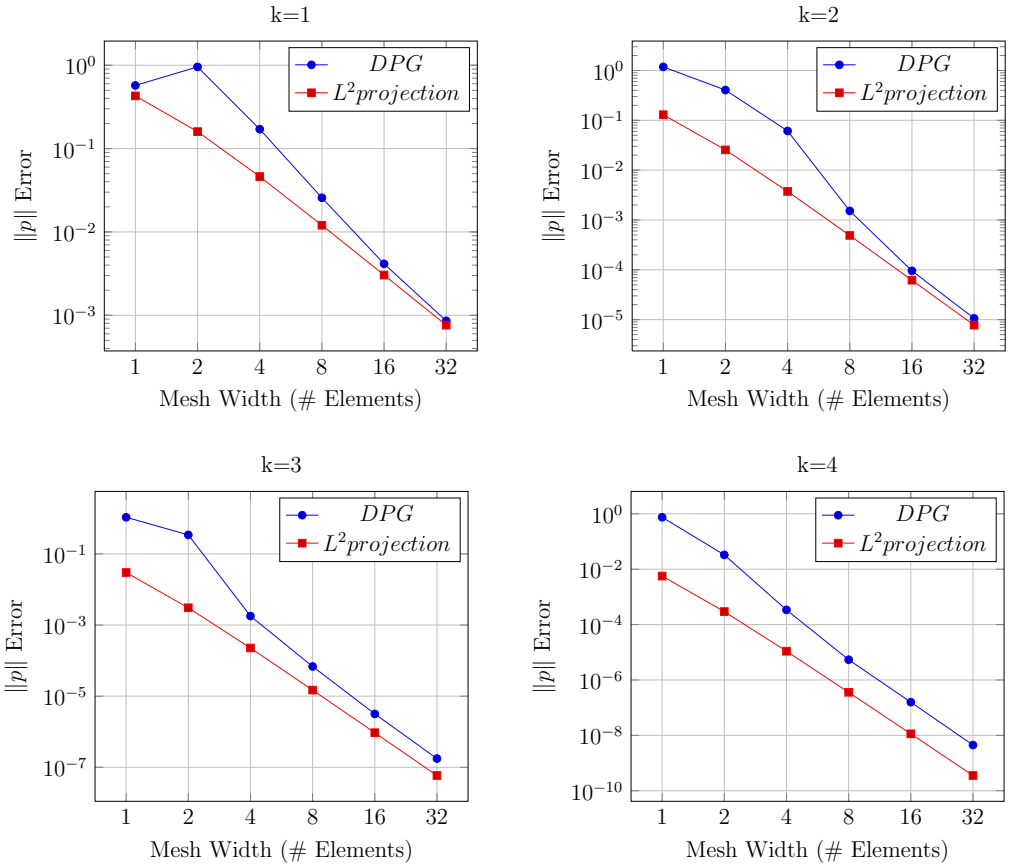


Figure 8.2: Kovasznay flow results for the DPG Navier-Stokes formulation using the standard graph norm with Re = 40: pressure error.

8.4—recall that for the solution velocity to lie in H^1, we must approximate the discontinuous boundary conditions from the classical problem; we do so by introducing a small "ramp" in the boundary conditions, interpolating between the no-flow condition at the wall and the unit velocity on the lid. For Navier-Stokes in two dimensions, the flow departs further and further from the exact symmetry of the Stokes problem as the Reynolds number increases. We seek

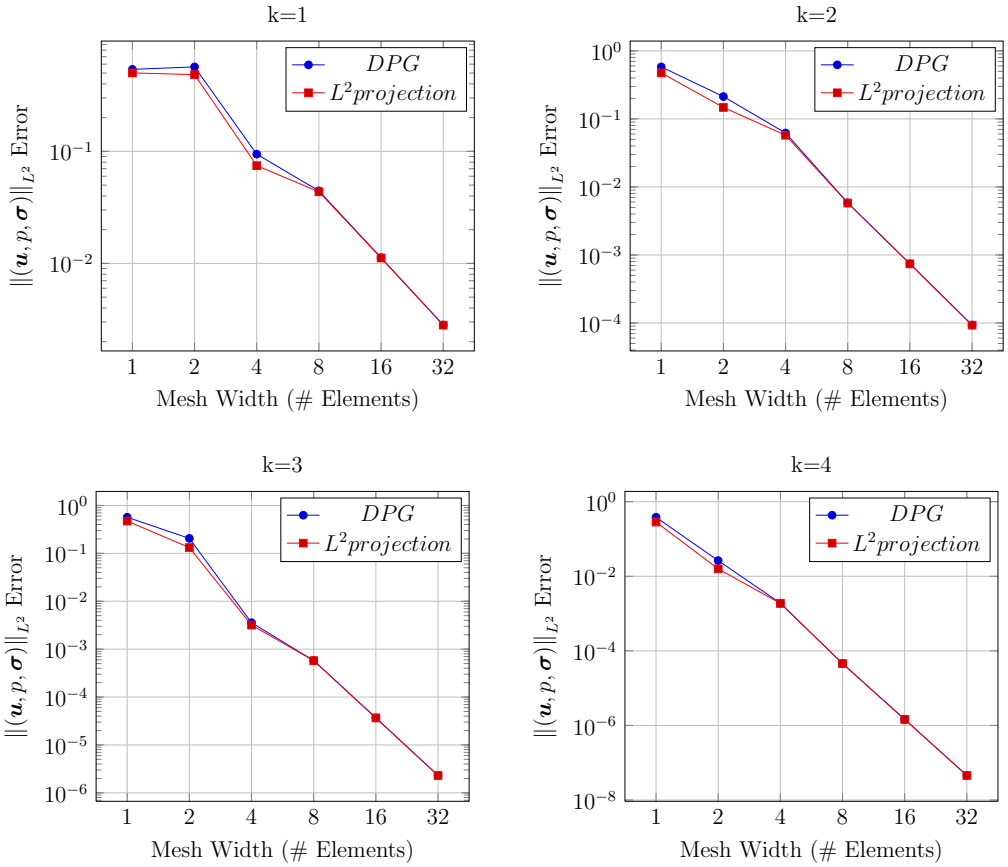


Figure 8.3: Kovasznay flow results for the DPG Navier-Stokes formulation using the standard graph norm with Re = 40: L^2 error of all field variables.

first of all to show that the method converges to solutions that qualitatively agree with accepted solutions in the literature; while we do present some careful (and encouraging) quantitative comparisons in the context of the Re = 1000 lid-driven cavity problem, such comparisons are not our principal objective.

Fixed Meshes. Starting with a zero initial guess, and using the VGP Navier-Stokes Formulation presented above with its (unscaled) adjoint graph norm, we solved the lid-driven cavity flow problem for Re = 100, 400, and 1000. The resulting streamline plots are shown in Figure 8.5.

We then solved each of these flow regimes using Navier-Stokes with adaptivity. We used the energy error of the linearized problem to drive refinements according to our usual greedy strategy with a threshold $\theta = 20\%$. In each case, we started with a 2×2 quartic mesh and performed six refinement steps. On the coarsest mesh, we started from a zero initial guess; for subsequent meshes, we used the projection of the solution thus far onto the refined mesh. Since our solution is linear in the fluxes and traces, we did

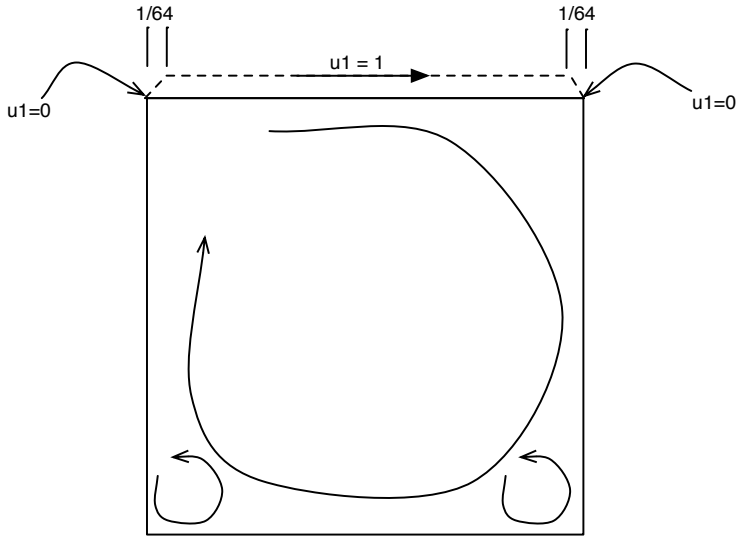
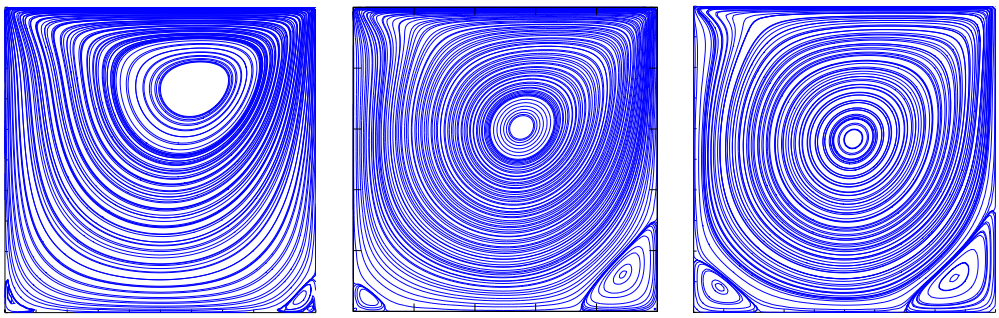


Figure 8.4: Sketch of lid-driven cavity flow. To obtain a solution in H^1, we linearly interpolate the lid boundary condition $u_1 = 1$ and the wall condition $u_1 = 0$ in segments of width $\frac{1}{64}$ on the left and right ends of the lid.

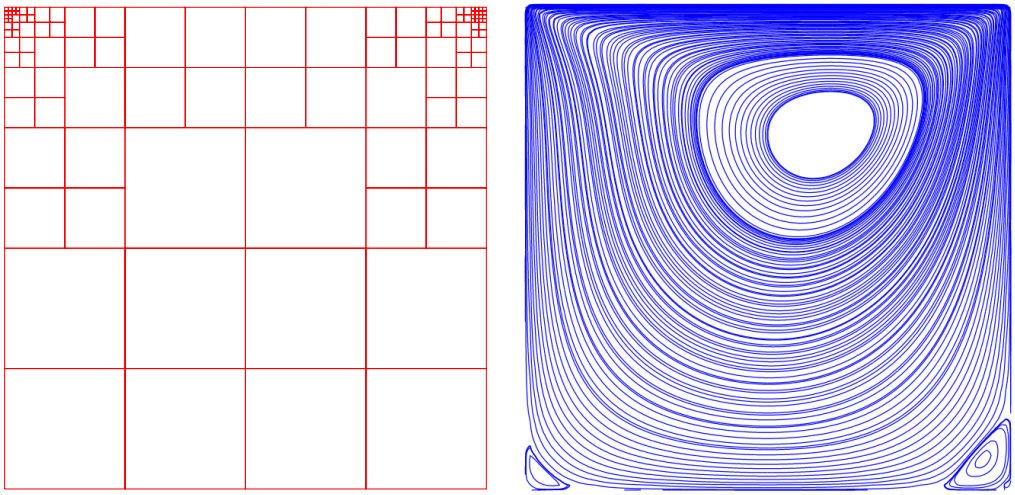
not accumulate in these, but instead solve for these at each Newton step. We continued Newton iterations on each refined mesh until the L^2 norm of the field variable increments—that is, $\left(\|\boldsymbol{u}\|^2 + \|p\|^2 + \|\boldsymbol{\sigma}\|^2\right)^{1/2}$—was less than a tolerance of 3×10^{-8}. On the final solve, we required a tighter tolerance of 3×10^{-9}. The resulting streamlines are shown in Figures 8.6, 8.7, and 8.8. The reported energy errors are for the linearized problem in the last Newton step.

For a Reynolds number of 5000, the strategy outlined above does not converge, presumably due to roundoff error in the solution of the optimal test functions or (more likely) in the solution of the global problem. To solve in this regime, we modified the strategy slightly: we used simple artificial time stepping with a fixed time step size of 1.0. Here, we relaxed the nonlinear tolerances to 10^{-3} for the intermediate solves, and 10^{-4} for the final solve. The results are shown in Figure 8.9.



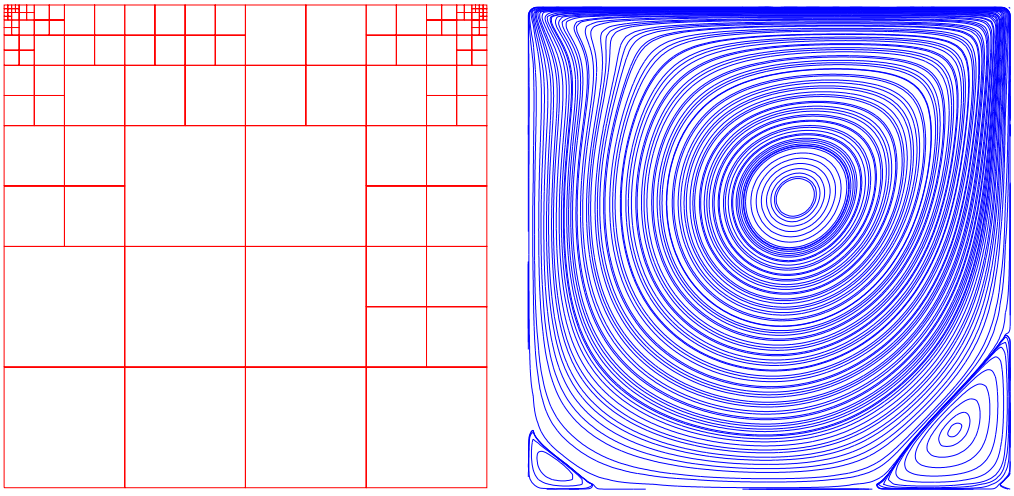
(a) Re = 100 (b) Re = 400 (c) Re = 1000

Figure 8.5: Streamlines for the cavity flow problem with Re = 100, 400, and 1000, each on a linear 64×64 mesh. We used $\Delta k = 2$ for the test space enrichment.



(a) mesh (b) streamlines

Figure 8.6: Streamlines for the cavity flow problem with Re = 100 on an adaptive quartic mesh after 6 refinements; final mesh has 109 elements (24,159 dofs). The energy error of the solution is 2.7×10^{-3}.

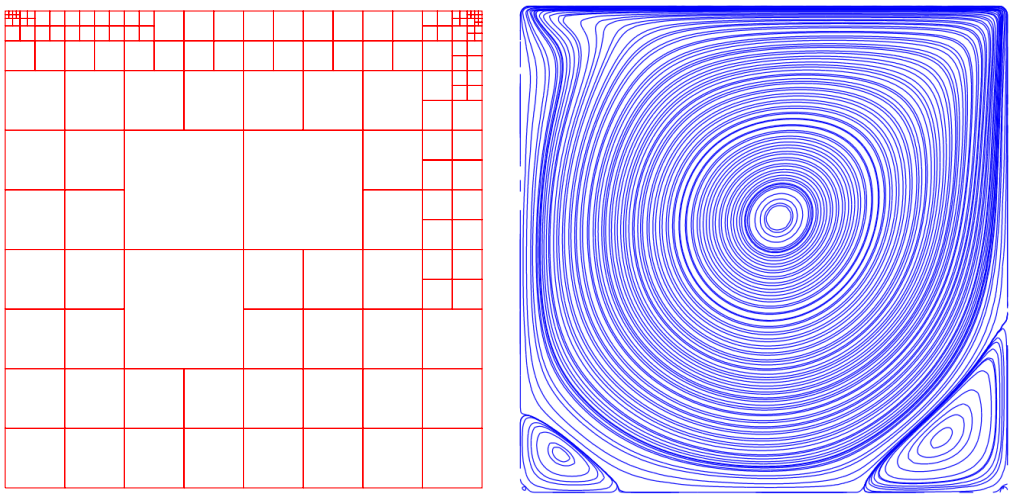


(a) mesh (b) streamlines

Figure 8.7: Streamlines for the cavity flow problem with Re = 400 on an adaptive quartic mesh after 6 refinements; final mesh has 109 elements (24,159 dofs). The energy error of the solution is 7.1×10^{-4}.

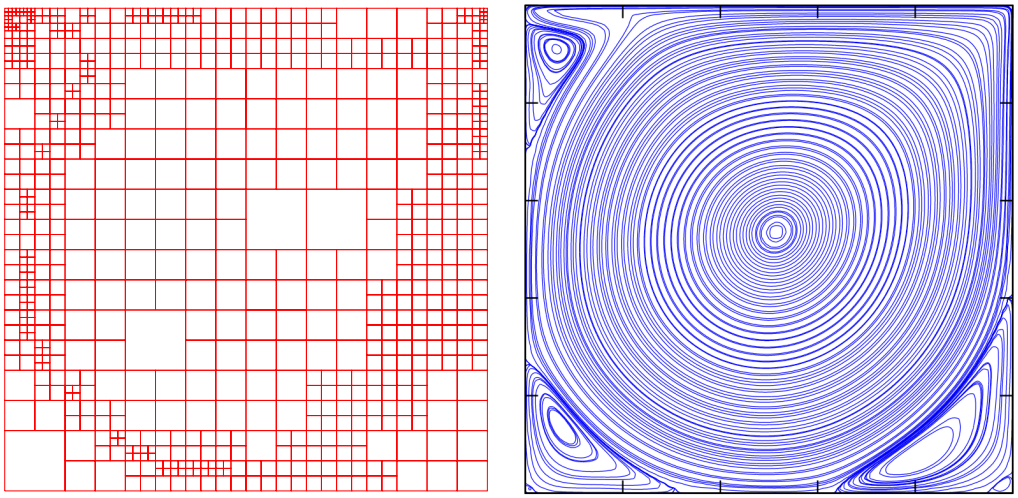
For the Re = 8000 case, we used an identical approach to the one we used for Re = 5000; the results are shown in 8.10. It is worth noting that somewhere between Re = 8000 and Re = 8050, the flow becomes oscillatory [18].

Despite the fact that the flow is known to become oscillatory for Reynolds numbers above 8050, we have found that we can compute solutions for higher Reynolds numbers using our steady-state solver. For example, Figure 8.11 shows a solution with Reynolds number of 10000, computed just as in the Re = 8000 case. The streamlines appear quite similar to plots of the transient solution provided by Bruneau and Saad [18]. We are unsure of the best interpretation of this solution; perhaps it is an unstable steady state—the



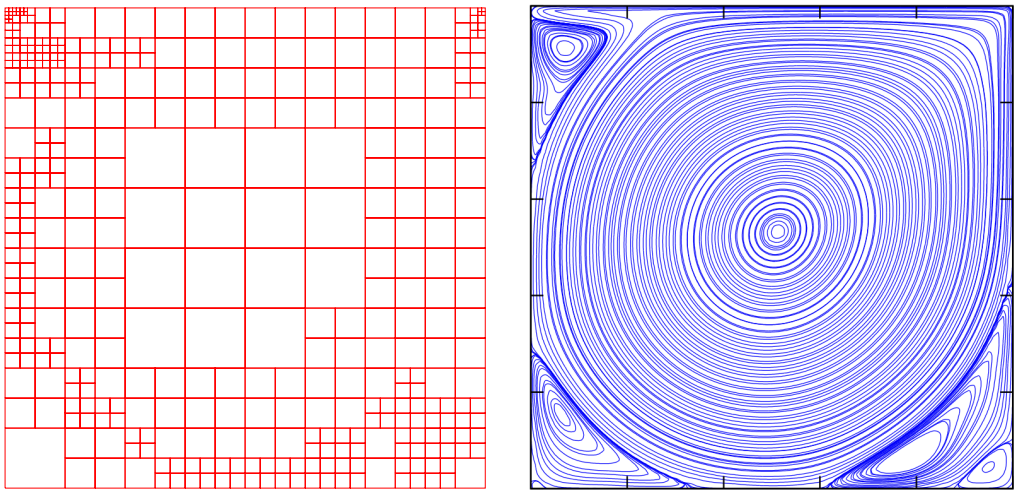
(a) mesh (b) streamlines

Figure 8.8: Streamlines for the cavity flow problem with Re = 1000 on an adaptive quartic mesh after 6 refinements; final mesh has 148 elements (32,686 dofs). The energy error of the solution is 2.9×10^{-4}.



(a) mesh (b) streamlines

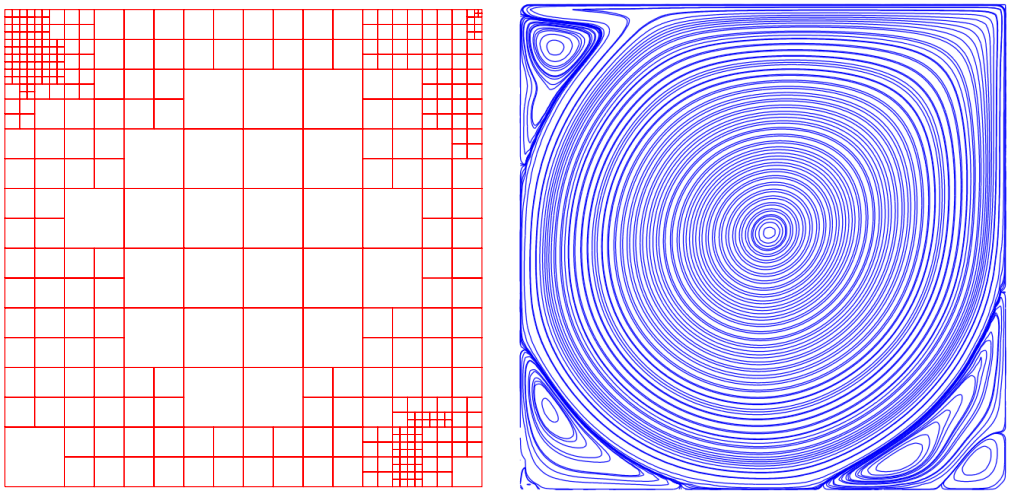
Figure 8.9: Streamlines for the cavity flow problem with Re = 5000 on an adaptive quartic mesh after 6 refinements and using artificial time stepping with dt = 1.0; final mesh has 763 elements (166,001 dofs). The energy error of the solution is 1.6×10^{-4}.



(a) mesh (b) streamlines

Figure 8.10: Streamlines for the cavity flow problem with Re = 8000 on an adaptive quartic mesh after 6 refinements and using artificial time stepping with dt = 1.0; final mesh has 400 elements (87,302 dofs). The energy error of the solution is 2.6×10^{-4}.

fact that we are able to resolve to an energy error of 3.0×10^{-4} does seem to suggest that this is a solution to the problem.



(a) mesh (b) streamlines

Figure 8.11: Streamlines for the cavity flow problem with Re = 10000 on an adaptive quartic mesh after 6 refinements and using artificial time stepping with dt = 1.0; final mesh has 379 elements (82,549 dofs). The energy error of the solution is 3.0×10^{-4}. Note that this is a steady state solution in a transient regime.

For numerical verification of our cavity flow solutions, we compare our Re = 1000 results with the extensive pseudo-spectral benchmark solution computed by Botella and Peyret [9]. We have computed many of the values reported by Botella and Peyret, and found that in every case our refined solutions agree well (in many cases, to three or four digits) with their reported numbers. In the interest of brevity, here we report a relevant subset of these results.

Perhaps the benchmark values of greatest engineering interest are the

extremal velocities along the horizontal and vertical centerlines. Table 8.1 shows these values for a series of h-refinements on a quartic mesh, together with the benchmark values. As can be seen, even the coarsest mesh has values not too far from the benchmark, and as we refine the values approach those of the benchmark.

Also of interest are the center locations of the various vortices and the vorticity and the streamfunction values at those points. The vortex centers are local extrema of the streamfunction—the center of the primary vortex is a local maximum, the centers of secondary vortices are local minima, the centers of tertiary vortices are again local maxima, and so on. The values for the primary vortex center are shown in Table 8.2. Here, only the vorticity value is off significantly in the coarsest 2×2 mesh, and after just one refinement, all values agree with the benchmark in the first digit. By the time we have completed 8 refinements, ψ agrees to four digits, while ω agrees to three. It is worth noting, however, that in several cases, our solver differs in some of the apparently converged digits from those reported by Botella and Peyret—e.g., our $u_{\max}$ appears to have converged to four or five digits, and the fourth digit differs from that in [9]. This may simply reflect the difference in our respective boundary conditions—recall that we approximate the discontinuity at the top corners, while Botella and Peyret solve with discontinuous boundary conditions. Alternately, the discrepancy may reflect accuracy limitations of the solution in [9]; in any case, the difference warrants further investigation.

Ref. Type	# Refs	dofs	$u_{\max}$	$y_{\max}$	$v_{\max}$	$x_{\max}$	$v_{\min}$	$x_{\min}$
Benchmark	—	—	0.3885698	0.1717	0.3769447	0.8422	-0.5270771	0.0908
h, $k = 4$	0	934	0.396339	0.242442	0.39888	0.812133	-0.478214	0.131708
h, $k = 4$	1	3570	0.393731	0.197367	0.377642	0.825603	-0.49186	0.10713
h, $k = 4$	2	6885	0.384899	0.179395	0.369079	0.832126	-0.511888	0.0857587
h, $k = 4$	3	9561	0.385443	0.178651	0.371756	0.838819	-0.523973	0.0934616
h, $k = 4$	4	13555	0.386732	0.177283	0.373124	0.839166	-0.526561	0.0924524
h, $k = 4$	5	22182	0.387341	0.173529	0.37668	0.841985	-0.528125	0.091373
h, $k = 4$	6	32027	0.388339	0.171578	0.376706	0.842174	-0.527996	0.0905222
h, $k = 4$	7	47763	0.38832	0.171756	0.376777	0.842211	-0.527	0.0907992
h, $k = 4$	8	62960	0.388328	0.17176	0.376816	0.842126	-0.526955	0.0908152

Table 8.1: Extrema of the velocity along vertical and horizontal centerlines for cavity flow with Re = 1000. Benchmark values are the highest-fidelity results ($N = 160$ case) from Botella and Peyret [9].

Ref. Type	# Refs	dofs	ψ	ω	x	y
Benchmark	—	—	0.1189366	2.067753	0.4692	0.5652
h, $k = 4$	0	934	0.147864	6.41215	0.478689	0.595329
h, $k = 4$	1	3570	0.121863	2.2432	0.465454	0.568378
h, $k = 4$	2	6885	0.119	2.03875	0.468219	0.561995
h, $k = 4$	3	9561	0.119314	2.059	0.471577	0.564492
h, $k = 4$	4	13555	0.119715	2.06886	0.471538	0.563466
h, $k = 4$	5	22182	0.119161	2.06428	0.469261	0.564757
h, $k = 4$	6	32027	0.119052	2.06538	0.469334	0.564946
h, $k = 4$	7	47763	0.11896	2.06663	0.4692	0.5652
h, $k = 4$	8	62960	0.118957	2.0666	0.4692	0.5652

Table 8.2: Location of the primary vortex center, and values of streamfunction and vorticity for Re = 1000. Benchmark values are the highest-fidelity results ($N = 160$ case) from Botella and Peyret [9].

8.2.3 Backward-Facing Step

We now return to the backward-facing step problem introduced in the context of the Stokes numerical experiments in Section 6.3. Following Biswas et al. [6], we consider the backward-facing step with a parabolic inlet velocity profile and the 1.9423 expansion we already used in the context of Stokes in Section 6.3. Using this geometry, Armaly showed experimentally that such a flow remains two-dimensional up to a Reynolds number of about 400 [2]. Biswas et al. note that Alleborn et al. [1] established the existence of a series of Moffatt eddies for vanishing Reynolds numbers. They note with some surprise[1] that their numerical experiments show a second Moffatt eddy (of width approximately .02) even for the Re = 1 case.[2] While we have reproduced the larger-scale features of their results for Reynolds numbers in the range $1 \leq \text{Re} \leq 100$, we have not been able to resolve a second Moffatt eddy either for Stokes or for the smaller Reynolds number. Curiously, we do resolve second Moffatt eddies for the Re = 10 and Re = 100 cases.

As with the cavity flow problem, our present objective is to demonstrate that, in the context of this difficult problem, DPG adaptively converges to a solution that agrees with the large-scale qualitative features of solutions previously reported in the literature—in this case, the solutions given by

[1] In particular, see their conclusions in [6].

[2] It is perhaps worth mentioning that Biswas et al. employ Reynolds numbers relative the length scale of the "hydraulic diameter," which amounts to double the height of the inlet. We follow their definition in the present discussion, though we have found it more convenient to use the inlet height as the defining length scale—hence our computations define a Reynolds number equal to half the Biswas et al. value.

Biswas et al. [6]. In the future, we plan to perform quantitative comparisons of, e.g., the length of the recirculation region. Similarly, we do not here perform any sensitivity analysis with regard to the length of the outflow region; we plan to do so in the future.

Reynolds 1 case. Here, we describe our experimental setup for $\mathrm{Re} = 1$. As in our efforts to resolve the Stokes cavity Moffatt eddies, we use our standard adaptivity, supplemented by some induced refinements in the recirculation region—this is necessary here as in cavity flow because the smaller Moffatt eddies are extremely weak phenomena that may not be picked up by our energy error computation until the mesh is very fine. In particular, for the first eight refinement steps, we induce refinements in the corner element and its siblings in the mesh hierarchy: in the first step this is two elements, owing to the way we constructed our mesh; in subsequent steps it is the four elements nearest the recirculating corner. We performed 14 refinement steps to obtain a 266-element mesh with 58,700 degrees of freedom, shown in Figure 8.12. We used as a nonlinear stopping criterion that the Euclidean combination of the L^2 norms of the field variables in the incremental solution was less than 10^{-4}. The energy error of the final incremental solution was 1.97×10^{-3}, an indication that our discretization has resolved the solution well. The streamlines shown in Figure 8.13, however, show no second Moffatt eddy.

We are unsure what to make of this result. It is worth noting that we chose to do eight induced refinements in the recirculating corner precisely

for the reason that this would give us elements of width $\frac{1}{128} = 7.8125 \times 10^{-3}$, below the .02 width of the supposed second Moffatt eddy. Our tentative idea, given our efforts to resolve the solution, is that Biswas et al. were mistaken, that the recirculation region they identified was a numerical artifact of some kind.

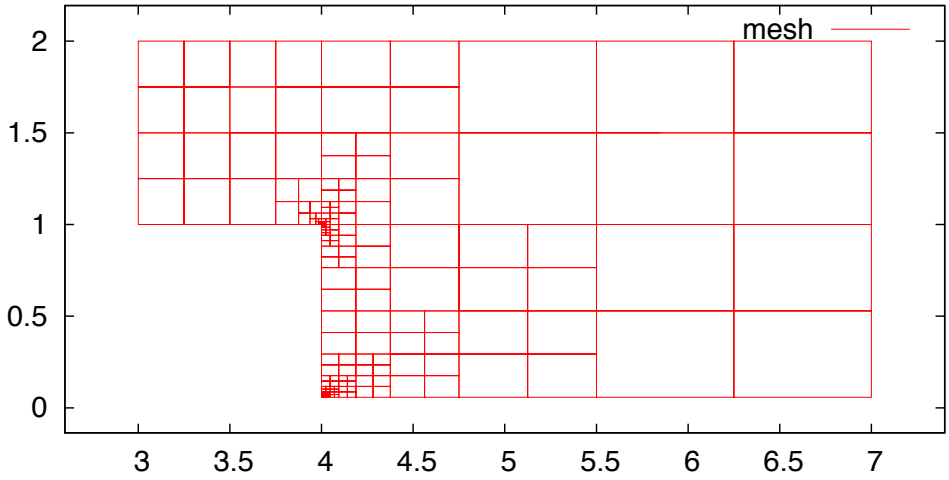


Figure 8.12: Mesh for the backward-facing step problem with Re = 1 on an adaptive quartic mesh after 14 refinements.

Reynolds 10 case. For Re = 10, we again used our standard adaptivity, supplemented by some induced refinements in the recirculation region. We performed 7 refinement steps to obtain a 149-element mesh with 32,939 degrees of freedom, shown in Figure 8.14. We used as a nonlinear stopping criterion that the Euclidean combination of the L^2 norms of the field variables in the incremental solution was less than 3×10^{-8}. The streamline plots in Figure 8.15 clearly show both the first and the second Moffatt eddy.

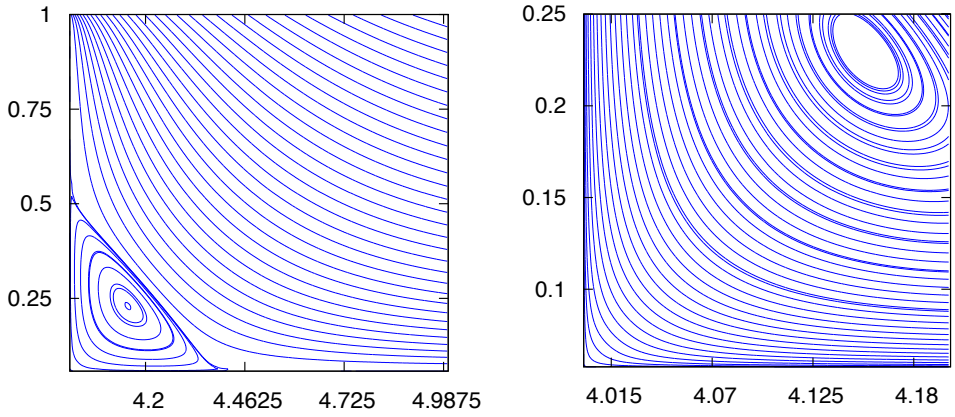


Figure 8.13: Streamlines for the backward-facing step problem with Re = 1 on an adaptive quartic mesh after 14 refinements; final mesh has 266 elements (58,700 dofs). The energy error of the solution is 1.97×10^{-3}. Biswas et al. [6] show a second Moffatt eddy of width approximately .02 in the lower left corner.

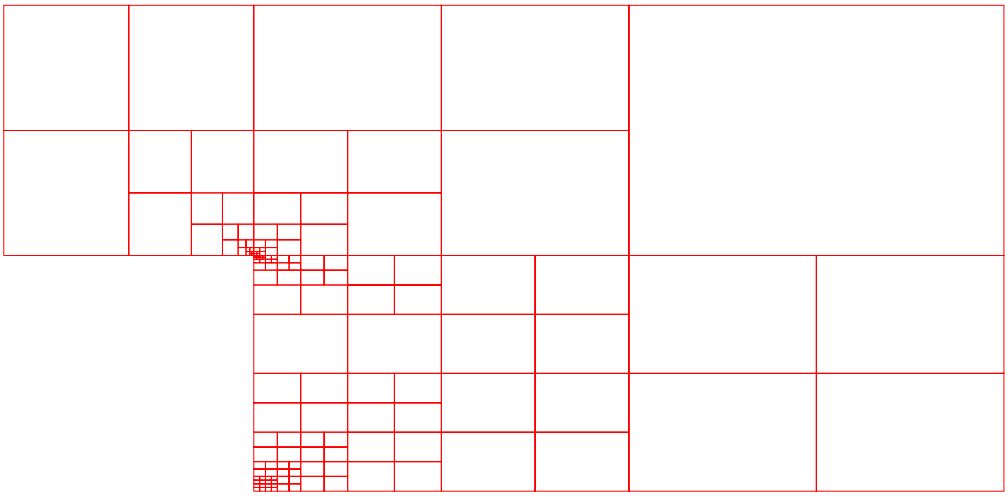


Figure 8.14: Mesh for the backward-facing step problem with Re = 10 on an adaptive quartic mesh after 7 refinements.

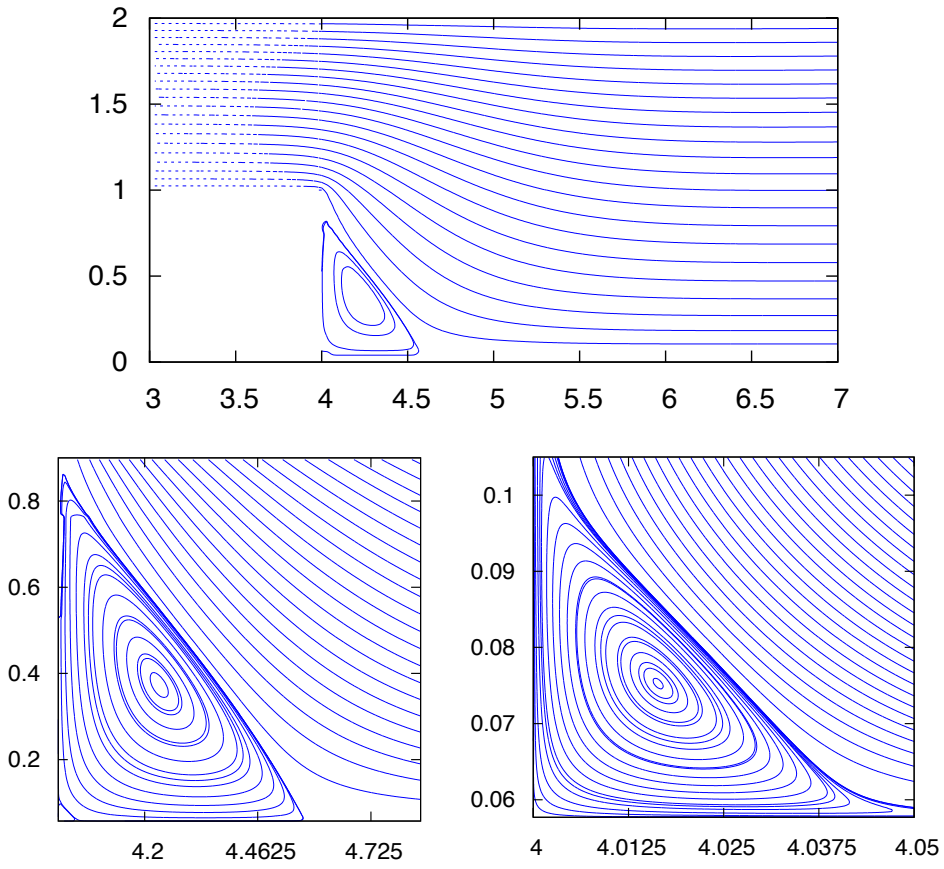


Figure 8.15: Streamlines for the backward-facing step problem with Re = 10 on an adaptive quartic mesh after 7 refinements; final mesh has 149 elements (32,939 dofs). The energy error of the solution is 1.27×10^{-2}.

Reynolds 100 case. For Re = 100, we again used our standard adaptivity, supplemented by some induced refinements in the recirculation region. We performed 7 refinement steps to obtain a 197-element mesh with 43,403 degrees of freedom, shown in Figure 8.16. We used as a nonlinear stopping criterion that the Euclidean combination of the L^2 norms of the field variables in the incremental solution was less than 3×10^{-8}. The streamline plots in Figure 8.17 clearly show both the first and the second Moffatt eddy.

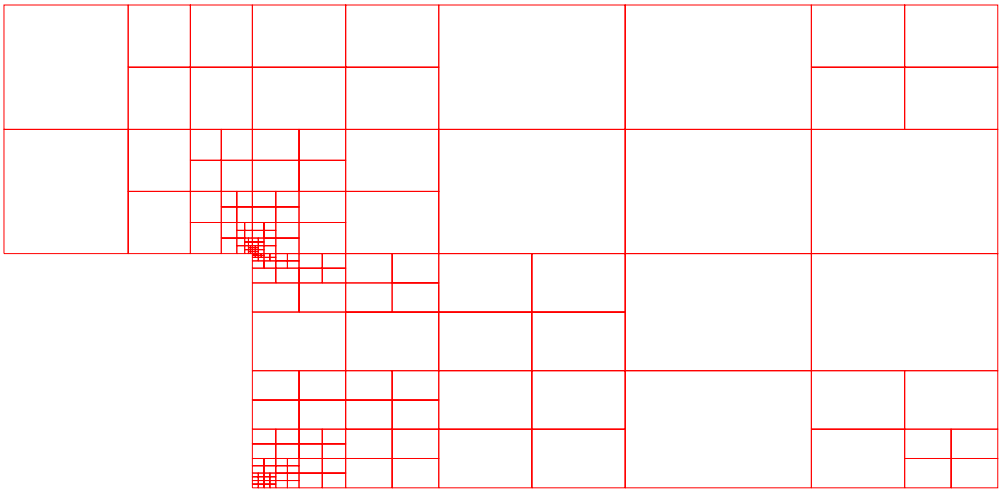


Figure 8.16: Mesh for the backward-facing step problem with Re = 100 on an adaptive quartic mesh after 7 refinements.

8.2.4 Flow Around a Cylinder

Another common model problem for the Navier-Stokes equations is flow past a cylinder. For Reynolds numbers between 6 and 40, a steady state solution exists with standing vortices in the cylinder wake. At some critical Reynolds number Re_c above 40, the flow becomes unsteady. Kovasznay

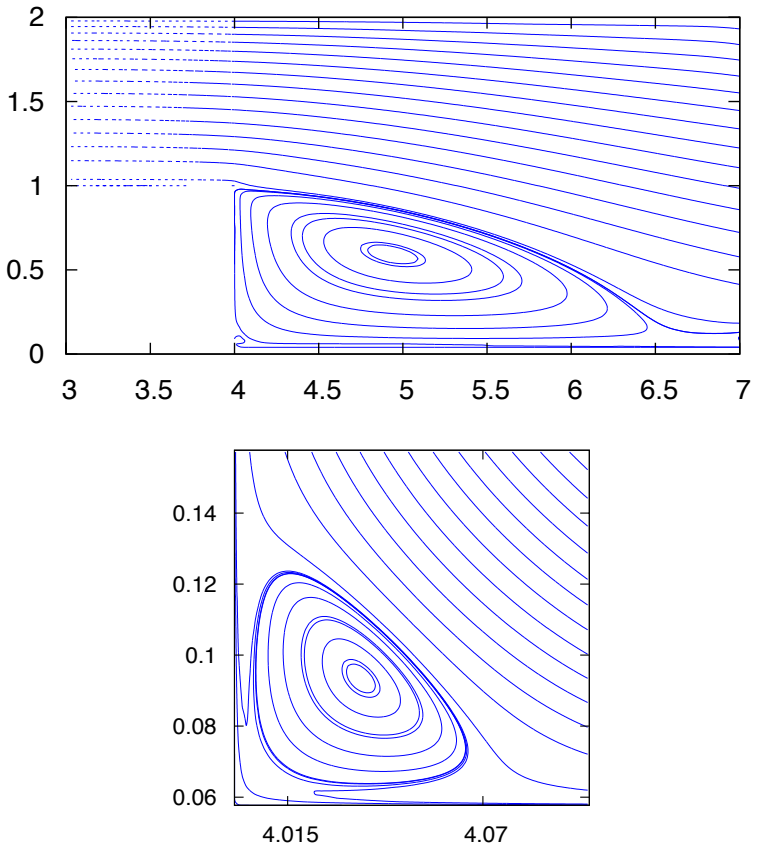


Figure 8.17: Streamlines for the backward-facing step problem with $\mathrm{Re} = 100$ on an adaptive quartic mesh after 7 refinements; final mesh has 197 elements (43,403 dofs). The energy error of the solution is 1.22×10^{-3}.

performed some early experiments, suggesting $\mathrm{Re}_c = 40$ [61]; recently, Kalita and Sen have argued, by a combination of numerical simulations and lab experiments, that $46.5 < \mathrm{Re}_c \leq 47$; they estimate $\mathrm{Re}_c \approx 46.5$ [57].

As with our previous experiments, here our goal is to examine the qualitative agreement of our adaptive solutions to the problem with those previously reported in the literature. While we do perform quantitative comparison for the drag coefficient, we do not perform any sensitivity analysis for this value with regard to the size of the domain.

The geometry of the problem is as shown in Figure 8.18; we take the domain to be a large enough box relative to the cylinder that free stream conditions may be assumed at the inflow and top and bottom of the domain. The Reynolds number is defined as

$$\mathrm{Re} = \frac{U_\infty D}{\nu}$$

for a free stream velocity U_∞, cylinder diameter D, and viscosity ν.

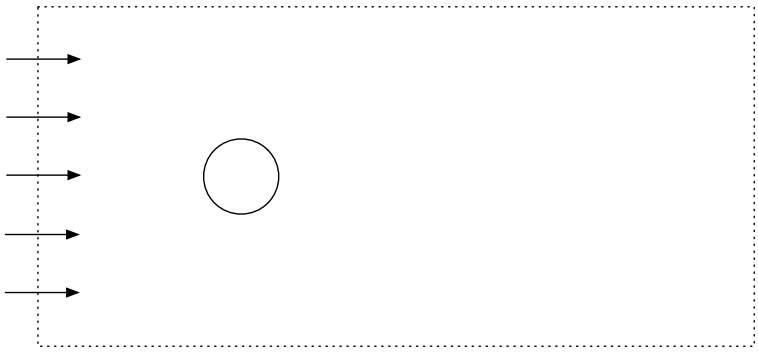


Figure 8.18: Schematic of the flow past a cylinder problem.

Following Kalita and Sen, we take the free stream velocity to be constant: $U_\infty = 1$. We accordingly impose Dirichlet boundary conditions on the velocity trace $\widehat{\boldsymbol{u}} = \binom{U_\infty}{0}$ at the inflow and top and bottom of the domain; we impose zero-traction ("do-nothing") conditions $\widehat{\boldsymbol{t}}_n = 0$ on the outflow.

Some statistics of interest for the problem are the drag coefficient c_D and the lift coefficient c_L—because of the symmetries inherent in the problem, we know a priori that c_L should be 0; computing it allows us to test our experiment. We can define drag and lift forces

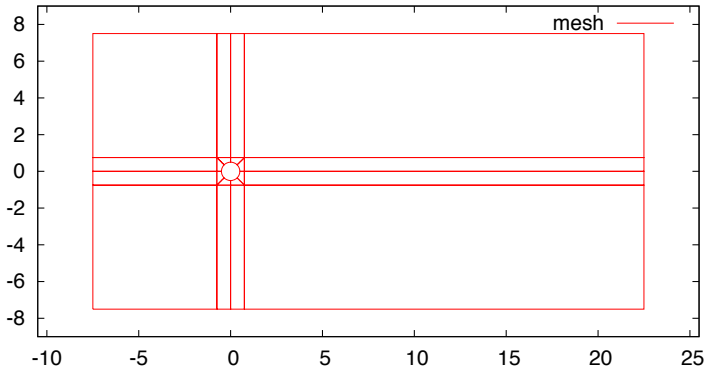
$$F_D = \int_S f_{\text{friction}} n_x - p n_x, \quad F_L = \int_S -(f_{\text{friction}} n_x + p n_y),$$

where the integral is around the circle S, $\boldsymbol{n} = \binom{n_x}{n_y}$ is the outward normal, and $f_{\text{friction}} \stackrel{\text{def}}{=} (\boldsymbol{\sigma}\boldsymbol{n}) \times \boldsymbol{n}$. The drag and lift coefficients are then defined as

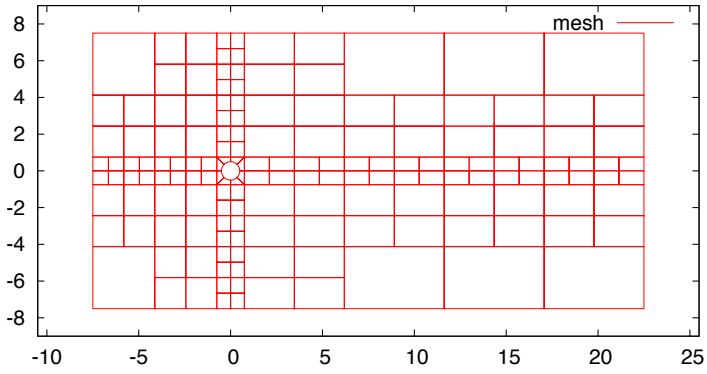
$$c_D = \frac{F_D}{\frac{1}{2}\rho U_\infty^2 D}, \quad c_L = \frac{F_L}{\frac{1}{2}\rho U_\infty^2 D}.$$

Thus, for our computation, in which we have non-dimensionalized in a manner that gives us unit-valued U_∞, D, and ρ, the values are simply $c_D = 2F_D$ and $c_L = 2F_L$. For the $\text{Re} = 40$ case, Kalita and Sen report the drag coefficients computed by several studies in the literature, including their own.

For our discretization, we took a preliminary twenty-element cubic mesh, shown in Figure 8.19(a), and performed some preliminary refinements to ensure that each element had an aspect ratio less than 2 while maintaining one-irregularity, giving us the mesh in Figure 8.19(b), which has 256 elements and 23,488 degrees of freedom.



(a) preliminary mesh

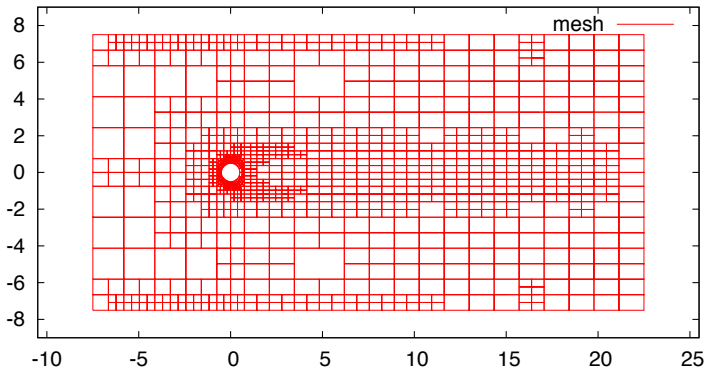


(b) initial mesh

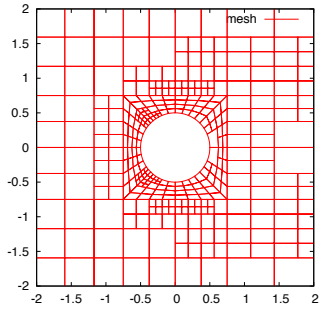
Figure 8.19: Preliminary cubic mesh for the flow past a cylinder problem at $\mathrm{Re} = 40$, and the mesh after some initial refinements were done to ensure that the mesh is roughly isotropic.

We then performed a series of four h-refinement steps according to our usual greedy refinement strategy. The final mesh is shown in Figure 8.20. The final mesh had 1,126 elements and 162,900 degrees of freedom. The energy error of the final linearized solution was 2.3×10^{-4}. The vorticity contours can be seen in Figure 8.21(a); the streamlines can be seen in Figure 8.21(b). These plots agree very well with those published by Kalita and Sen in [57].

With each refinement, we computed the drag and lift coefficients; the results are listed in Table 8.3. The lift coefficient is very near zero in all computations, and the drag coefficient appears to be converging to a value near 1.674. This value is somewhat above the value of 1.590 reported by Kalita & Sen, and their value is itself above values reported earlier in the literature. Examining the refinement pattern in Figure 8.20, we note that there are refinements at the top and bottom boundaries of the domain; these suggest that our Dirichlet conditions on both velocity components may have been too strong. We therefore hypothesize that it is for this reason that our drag coefficient differs from values previously reported. In the future, we plan to repeat the experiment with other boundary conditions, as well as to perform sensitivity analysis with regard to the size of the domain. We also plan to examine other quantities of interest, including the angle at which the standing vortex attaches to the cylinder and the length of the recirculation region.

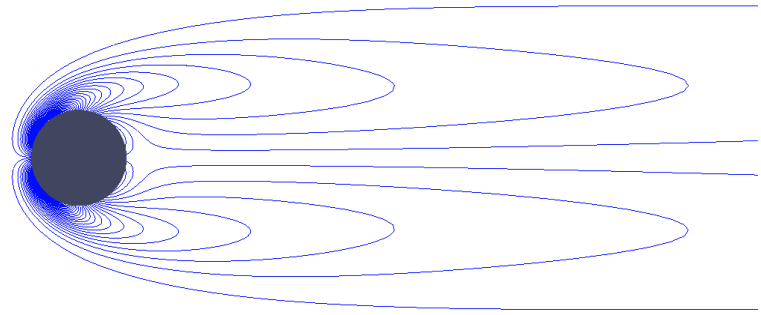


(a) final mesh

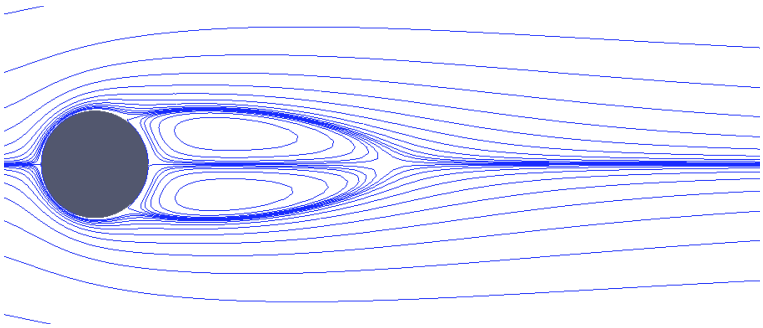


(b) final mesh, detail

Figure 8.20: Cubic mesh for the flow past a cylinder problem at Re = 40: mesh after 4 refinements, and a detail of that mesh.



(a) vorticity



(b) stream function

Figure 8.21: Vorticity and stream function contours for the cylinder flow problem with $k = 3$ after 4 refinements.

Mesh Type	# Refinements	dofs	c_D	c_L
Kalita & Sen [57]	-	181×301 grid	1.590	-
DPG, $k = 3$	0	23,488	1.62179	4.28×10^{-14}
DPG, $k = 3$	1	32,232	1.67476	-6.06×10^{-13}
DPG, $k = 3$	2	50,476	1.67407	-1.93×10^{-13}
DPG, $k = 3$	3	86,980	1.67400	-5.32×10^{-12}
DPG, $k = 3$	4	162,900	1.67387	1.14×10^{-13}

Table 8.3: Drag and lift coefficients computed for the flow past a cylinder problem. The lift coefficients are provided as verification of the code; the lift can be shown to be zero analytically.

Chapter 9

Conditioning Studies

A question that should be asked of any numerical method is how susceptible it is to roundoff error. One way of measuring this susceptibility in the context of matrix systems, is by means of the (2-norm) condition number of the matrix, defined for a matrix A by

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)},$$

where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are the maximum and minimum eigenvalues of A. The condition number arises explicitly in proofs of the convergence of iterative solvers such as the conjugate gradient method—bounds for the number of iterations to converge are put in terms of the condition number of the matrix. Although the dependence for direct solvers is less direct, nevertheless the accuracy of a direct solve does in general depend on the conditioning of the problem. A rule of thumb often used is that for a matrix with condition number of 10^d, one may lose up to d digits of accuracy.

Thus, in this chapter we conduct a few studies of conditioning in the context of DPG and Camellia. This is meant merely as a first foray into a substantial area of study. Ultimately, such explorations may yield better understanding of how best to precondition the DPG system matrix

and perhaps also better understanding of numerics implied by the test norm selection.

We begin in Section 9.1 with an examination of the conditioning properties of the nodal bases provided to Camellia by Intrepid, and compare these with the hierarchical Lobatto bases which we have added to Camellia for the purpose. We then turn our attention to the conditioning of the local and global solves for the DPG VGP Stokes system; in Section 9.2 we consider the graph norm, standard in most of our experiments, and in Section 9.3 we consider some variants of this norm and their effect on conditioning. In Section 9.4, we examine the effect of using Lobatto bases in the test space on the conditioning of the DPG matrices, and in Section 9.5 we look at the effect of static condensation on the conditioning of the DPG stiffness matrix.

9.1 Basis Conditioning Studies

To study the conditioning properties of a finite element basis $\{e_i\}$, the standard practice is to evaluate the condition number of the mass matrix and the stiffness matrix, where the mass matrix is defined by $A_{ij} = (e_i, e_j)_{L^2}$, and the stiffness matrix is $K_{ij} = (De_i, De_j)_{L^2}$, D being the differential operator associated with the basis—gradient, curl, or divergence. Because the null space of D is non-empty, the matrix K will have some zero eigenvalues. For the purpose of studying conditioning, these are neglected: the condition number is taken to be the ratio between the maximum and the minimum nonzero eigenvalues. Of particular interest for higher-order methods such as DPG is

the growth of the condition numbers as polynomial order increases.

It is worth emphasizing the way that we expect these condition numbers to enter our DPG computations. For the local solves for the optimal test functions, there are contributions corresponding to both mass and stiffness matrix entries of the test basis. In these local solves, the bilinear form, which involves the trial space basis, enters on the right hand side. We can then form the global system matrix based just on the optimal test coefficients and the test space Gram matrix (see Section 2.1.6). Because we do not take any derivatives of the trial space basis, it is evident that the conditioning of the global system is independent of the conditioning of the stiffness matrix on the trial space. How the global system's conditioning depends on the test basis is less clear, and part of the rationale for the present study.

9.1.1 Intrepid Basis

Trilinos's Intrepid package provides nodal H^1-, $H(\text{div})$-, and $H(\text{curl})$-conforming bases. One can select either equispaced nodes or *spectral* nodes, where the spectral nodes are reputed to offer better conditioning properties. In Camellia, we therefore use the spectral nodes exclusively. We compute the condition numbers as described above for polynomial orders $k = 1$ to 20, on a single-element, unit square mesh. We perform diagonal (Jacobi) scaling of each matrix prior to computing the condition numbers—the idea being that many preconditioners for iterative methods will perform such a step, and Gaussian elimination will do this implicitly.

The results are shown in Table 9.1. As can be seen, the growth in mass matrix condition numbers is extremely modest for both H^1 and $H(\text{div})$ (throughout this study, we omit the numbers for $H(\text{curl})$, as these are identical in 2D to those for $H(\text{div})$). The growth in the stiffness matrix condition numbers is perhaps of more concern—particularly when one considers that, for instance, condition numbers of finite element matrices for second-order elliptic boundary value problems generally scale as $O(h^{-2})$ [14, p. 261].

Polynomial Order	H^1 stiffness	H^1 mass	$H(\text{div})$ stiffness	$H(\text{div})$ mass
1	1.50E+00	9.00E+00	1.00E+00	3.00E+00
2	3.43E+00	9.11E+00	2.72E+00	3.02E+00
3	1.13E+01	9.26E+00	7.73E+00	3.04E+00
4	2.68E+01	9.41E+00	1.67E+01	3.07E+00
5	4.86E+01	9.55E+00	3.05E+01	3.09E+00
6	7.69E+01	9.69E+00	5.04E+01	3.11E+00
7	1.12E+02	9.81E+00	7.77E+01	3.13E+00
8	1.55E+02	9.93E+00	1.15E+02	3.15E+00
9	2.06E+02	1.00E+01	1.64E+02	3.17E+00
10	2.66E+02	1.01E+01	2.27E+02	3.18E+00
11	3.35E+02	1.02E+01	3.08E+02	3.20E+00
12	4.14E+02	1.03E+01	4.08E+02	3.21E+00
13	5.03E+02	1.04E+01	5.31E+02	3.23E+00
14	6.03E+02	1.05E+01	6.79E+02	3.24E+00
15	7.14E+02	1.06E+01	8.54E+02	3.25E+00
16	8.37E+02	1.06E+01	1.06E+03	3.26E+00
17	9.72E+02	1.07E+01	1.30E+03	3.27E+00
18	1.12E+03	1.08E+01	1.58E+03	3.28E+00
19	1.28E+03	1.08E+01	1.89E+03	3.29E+00
20	1.46E+03	1.09E+01	2.25E+03	3.30E+00

Table 9.1: Condition numbers for Intrepid H^1 and $H(\text{div})$ bases on the quadrilateral: stiffness and mass matrices. The condition numbers for $H(\text{curl})$ are identical to those for $H(\text{div})$.

9.1.2 Lobatto Basis

With the idea of improving on the conditioning in the Intrepid stiffness matrices, we have added to Camellia bases derived from the Lobatto (that

is, the integrated Legendre) polynomials—for present purposes, we restrict ourselves to H^1 and $H(\mathrm{div})$ bases on quadrilateral elements. Here, we briefly specify these bases. The Legendre polynomials can be computed recursively, according to the formula

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= x \\ L_n(x) &= \frac{2n-1}{n} x L_{n-1}(x) - \frac{n-1}{n} L_{n-2}(x). \end{aligned}$$

Thus specified, $\|L_n\|^2_{(-1,1)} = \frac{2}{2n+1}$. Moreover, the Legendre polynomials form an orthogonal basis for $L^2(-1,1)$. The Lobatto polynomials $\ell_n(x)$ are scaled integrals of these:

$$\ell_n(x) = \frac{1}{\|L_n\|} \int_{-1}^{x} L_n(x).$$

For a degree-n H^1 basis on the quadrilateral, we define

$$\phi_{ij}(x,y) = \ell_i(x)\ell_j(y) \quad i,j \in \{0,\dots,n\}.$$

We scale these by $\left(\|L_i\|^2 \|\ell_j\|^2 + \|\ell_i\|^2 \|L_j\|^2\right)^{1/2}$, so that the stiffness matrix has unit diagonal. For a degree-n $H(\mathrm{div})$ basis, we define for $i,j \in \{0,\dots,n\}$

$$\boldsymbol{\psi}_{ij} = \begin{cases} (\ell_i(x)L_j(y), 0) & \text{if } i = 0 \text{ and } j > 0 \\ (0, L_i(x)\ell_j(y)) & \text{if } i > 0 \text{ and } j = 0 \\ (\ell_i(x)L_j(y), L_i(x)\ell_j(y)) & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

Note that the $i = 0$ and $j = 0$ cases coincide exactly with the divergence-free subspace of the basis. This allows us to scale divergence-free members of the basis so that the mass matrix of this subspace has unit diagonal while

scaling the rest of the space so that its stiffness matrix has unit diagonal. What is more, for $i, j > 0$, $\nabla \cdot \boldsymbol{\psi}_{ij} = 2L_i(x)L_j(y)$, so that the divergence of this subspace inherits the L^2-orthogonality of the Legendre polynomials—all the non-zero entries of the $H(\text{div})$ stiffness matrix will lie along the diagonal, resulting in perfect conditioning.

As can be seen in Table 9.2, we do indeed observe a perfectly conditioned $H(\text{div})$ stiffness matrix, but the H^1 stiffness matrix is little better than that for Intrepid's nodal basis, and the mass matrices are both considerably worse-conditioned. The H^1 mass matrix is badly enough conditioned that, according to the rule of thumb mentioned above, we can expect to lose 7 digits, and this on a unit square domain. While this is striking and somewhat in tension with received wisdom to the effect that Lobatto bases offer the best conditioning properties, it is worth noting that similar results have previously been reported (see e.g. [75]).

9.2 Stokes Conditioning: Graph Norm

We now turn to a DPG formulation central to this dissertation: the Stokes velocity-gradient-pressure (VGP) formulation, with our usual graph norm. We construct the global system matrix and examine its condition numbers for meshes ranging from 2×2 to 16×16 elements with polynomial order $k = 1$ to 4. The results can be seen in Table 9.3. The condition numbers do become fairly large—as high as 10^7 for $k = 4$ on a 16×16 mesh, but they are growing at very close to the usual $O(h^{-2})$ rate.

Polynomial Order	H^1 stiffness	H^1 mass	$H(\text{div})$ stiffness	$H(\text{div})$ mass
1	1.00E+00	1.00E+00	1.00E+00	1.00E+00
2	2.20E+01	1.45E+03	1.00E+00	2.65E+01
3	2.20E+01	1.45E+03	1.00E+00	2.71E+01
4	6.52E+01	1.57E+04	1.00E+00	1.00E+02
5	6.52E+01	1.57E+04	1.00E+00	1.01E+02
6	1.33E+02	7.76E+04	1.00E+00	2.48E+02
7	1.33E+02	7.76E+04	1.00E+00	2.50E+02
8	2.25E+02	2.61E+05	1.00E+00	4.99E+02
9	2.25E+02	2.61E+05	1.00E+00	5.00E+02
10	3.44E+02	6.93E+05	1.00E+00	8.78E+02
11	3.44E+02	6.93E+05	1.00E+00	8.78E+02
12	4.90E+02	1.57E+06	1.00E+00	1.41E+03
13	4.90E+02	1.57E+06	1.00E+00	1.41E+03
14	6.63E+02	3.18E+06	1.00E+00	2.13E+03
15	6.63E+02	3.18E+06	1.00E+00	2.13E+03
16	8.65E+02	5.90E+06	1.00E+00	3.06E+03
17	8.65E+02	5.90E+06	1.00E+00	3.06E+03
18	1.10E+03	1.03E+07	1.00E+00	4.22E+03
19	1.10E+03	1.03E+07	1.00E+00	4.22E+03
20	1.36E+03	1.68E+07	1.00E+00	5.65E+03

Table 9.2: Condition numbers for Lobatto H^1 and $H(\text{div})$ bases on the quadrilateral: stiffness and mass matrices. The condition numbers for $H(\text{curl})$ are identical to those for $H(\text{div})$.

Order	Mesh Size	Condition #	Ratio
k=1	2×2	8.48E+03	
	4×4	5.48E+04	6.46
	8×8	2.61E+05	4.76
	16×16	1.26E+06	4.82
k=2	2×2	3.50E+04	
	4×4	1.82E+05	5.19
	8×8	8.16E+05	4.50
	16×16	3.73E+06	4.57
k=3	2×2	1.08E+05	
	4×4	4.96E+05	4.58
	8×8	2.13E+06	4.29
	16×16	9.34E+06	4.38
k=4	2×2	2.47E+05	
	4×4	1.09E+06	4.40
	8×8	4.61E+06	4.24
	16×16	1.98E+07	4.29

Table 9.3: Stokes VGP with the standard graph norm, global stiffness matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

Of course, to construct this system, we require a set of local solves for the optimal test functions on each element. The matrices for these systems are Gram (inner product) matrices for the test space norm. We examine the condition numbers of these matrices (note that because the material data is constant and the mesh uniform, the Gram matrix on each element will be identical). The results are listed in Table 9.4. Here, the largest condition numbers are slightly worse than those for the global solve, but the growth rates of the condition numbers are a greater concern: these are growing as $O(h^{-3})$ or $O(h^{-4})$. This motivates some experimentation with alternative test space norms, to which we turn in the next section.

Order	Mesh Size	Condition #	Ratio
k=1	2×2	4.82E+04	
	4×4	5.47E+05	11.36
	8×8	7.20E+06	13.15
	16×16	1.03E+08	14.38
k=2	2×2	1.30E+05	
	4×4	1.44E+06	11.07
	8×8	1.86E+07	12.86
	16×16	2.69E+08	14.49
k=3	2×2	2.84E+05	
	4×4	3.12E+06	11.01
	8×8	3.99E+07	12.76
	16×16	5.63E+08	14.12
k=4	2×2	5.75E+05	
	4×4	6.22E+06	10.82
	8×8	7.64E+07	12.29
	16×16	1.02E+09	13.34

Table 9.4: Stokes VGP with the standard graph norm, Gram matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

9.3 Stokes Conditioning: Graph Norm Variations

Recall that at the end of Chapter 6 we discussed a scaled graph norm, defined as

$$\begin{aligned}\|(\boldsymbol{v}, q, \boldsymbol{\tau})\|_V^2 :=& h^2 \|\nabla q + \nabla \cdot \boldsymbol{\tau}\|^2 + \|\nabla \cdot \boldsymbol{v}\|^2 + \|\boldsymbol{\tau} - \nabla \boldsymbol{v}\|^2 \\ &+ \left\|\frac{1}{h}\boldsymbol{v}\right\|^2 + \|q\|^2 + \|\boldsymbol{\tau}\|^2 . \end{aligned} \tag{9.3.1}$$

Since we are concerned with conditioning, we might think about the h-scales in the test space norm. Now, the Piola transform induces a $\frac{1}{h}$ scale on values in $H(\text{div})$ (i.e. $\boldsymbol{\tau}$) and derivatives of H^1 functions (i.e. ∇q, $\nabla \cdot \boldsymbol{v}$, and $\nabla \boldsymbol{v}$), and a $\frac{1}{h^2}$ scale for derivatives of $H(\text{div})$ functions (i.e. $\nabla \cdot \boldsymbol{\tau}$). Thus we observe that all the terms in the above norm scale as $\frac{1}{h}$, with the exception of the q terms, which have unit scale. We are free to rescale our test spaces as we like, so with an eye toward better Gram matrix conditioning, we rescale q by $\frac{1}{h}$, so that our test norm becomes:

$$\begin{aligned}\|(\boldsymbol{v}, q, \boldsymbol{\tau})\|_V^2 :=& \|\nabla q + h\nabla \cdot \boldsymbol{\tau}\|^2 + \|\nabla \cdot \boldsymbol{v}\|^2 + \|\boldsymbol{\tau} - \nabla \boldsymbol{v}\|^2 \\ &+ \left\|\frac{1}{h}\boldsymbol{v}\right\|^2 + \left\|\frac{1}{h}q\right\|^2 + \|\boldsymbol{\tau}\|^2 . \end{aligned}$$

Using this as our test norm, the results in Table 9.5 show that our Gram matrix condition numbers no longer grow as the mesh is refined, corroborating our reasoning about the h scales in the test norm. However, using this norm we observed considerably worse results in practice: we no longer achieved optimal convergence rates in manufactured solutions, for example. The explanation lies in the resulting condition numbers in the global system matrix, which are

shown in Table 9.6: there the condition numbers are growing at a rate around $O(h^{-4})$.

Order	Mesh Size	Condition #	Ratio
k=1	2×2	2.5E+03	-
	4×4	2.5E+03	1
	8×8	2.5E+03	1
	16×16	2.5E+03	1
k=2	2×2	5.6E+03	-
	4×4	5.6E+03	1
	8×8	5.6E+03	1
	16×16	5.6E+03	1
k=3	2×2	1.1E+04	-
	4×4	1.1E+04	1
	8×8	1.1E+04	1
	16×16	1.1E+04	1
k=4	2×2	2.0E+04	-
	4×4	2.0E+04	1
	8×8	2.0E+04	1
	16×16	2.0E+04	1

Table 9.5: Stokes VGP with the scaled graph norm and an additional $\frac{1}{h}$ scale on the q terms, Gram matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

Order	Mesh Size	Condition #	Ratio
$k = 1$	2×2	1.78E+03	-
	4×4	2.20E+04	12.3
	8×8	8.45E+05	38.5
	16×16	3.18E+07	37.7
$k = 2$	2×2	3.88E+03	-
	4×4	4.52E+04	11.7
	8×8	1.66E+06	36.8
	16×16	5.99E+07	36.0
$k = 3$	2×2	6.93E+03	-
	4×4	1.19E+05	17.2
	8×8	4.12E+06	34.6
	16×16	1.47E+08	35.6
$k = 4$	2×2	1.42E+04	-
	4×4	4.14E+05	29.2
	8×8	1.29E+07	31.1

Table 9.6: Stokes VGP with the scaled graph norm and an additional $\frac{1}{h}$ scale on the q terms, stiffness matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

With the idea that perhaps we asked for too much in pursuing minimal condition numbers in the Gram matrix, we instead try the scaled graph norm without the $\frac{1}{h}$ scaling in the q terms (i.e. we use exactly the norm given in Equation 9.3.1). Now our Gram matrix condition numbers, as shown in Table 9.7, scale approximately as $O(h^{-2})$. Remarkably, the global stiffness matrix condition numbers also scale as $O(h^{-2})$, as shown in Table 9.8.

Order	Mesh Size	Condition #	Ratio
k=1	2×2	4.82E+04	
	4×4	1.37E+05	2.84
	8×8	4.51E+05	3.29
	16×16	1.62E+06	3.60
k=2	2×2	1.30E+05	
	4×4	3.63E+05	2.79
	8×8	1.18E+06	3.25
	16×16	4.21E+06	3.57
k=3	2×2	2.84E+05	
	4×4	7.81E+05	2.75
	8×8	2.51E+06	3.22
	16×16	8.93E+06	3.55
k=4	2×2	5.75E+05	
	4×4	1.56E+06	2.71
	8×8	4.95E+06	3.18
	16×16	1.74E+07	3.52

Table 9.7: Stokes VGP with the scaled graph norm, Gram matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

9.4 Lobatto Bases: Effect on Gram and Stiffness Matrix Conditioning

Having examined above the conditioning of the Lobatto bases in isolation, we now consider their effect in practice, when we use them for the test space in the Stokes problem; here we continue to use the scaled graph norm. Tables 9.9 and 9.10 show the Gram and stiffness matrix results, respectively.

Comparing like entries with the previous results, we do see some improvement in the condition numbers.

Order	Mesh Size	Condition #	Ratio
k=1	2×2	1.67E+04	
	4×4	9.29E+04	5.56
	8×8	4.19E+05	4.51
	16×16	2.08E+06	4.96
k=2	2×2	7.26E+04	
	4×4	3.13E+05	4.32
	8×8	1.31E+06	4.17
	16×16	5.90E+06	4.52
k=3	2×2	2.14E+05	
	4×4	8.55E+05	3.99
	8×8	3.42E+06	4.00
	16×16	1.46E+07	4.28
k=4	2×2	4.97E+05	
	4×4	1.89E+06	3.81
	8×8	7.49E+06	3.96
	16×16	3.13E+07	4.18

Table 9.8: Stokes VGP with the scaled graph norm, global stiffness matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

Order	Mesh Size	Condition #	Ratio
$k = 1$	2×2	6.44E+03	-
	4×4	1.36E+04	2.11
	8×8	4.07E+04	2.99
	16×16	1.48E+05	3.63
$k = 2$	2×2	5.31E+04	-
	4×4	9.83E+04	1.85
	8×8	2.42E+05	2.46
	16×16	7.68E+05	3.18
$k = 3$	2×2	7.09E+04	-
	4×4	1.34E+05	1.88
	8×8	3.39E+05	2.53
	16×16	1.10E+06	3.26
$k = 4$	2×2	3.35E+05	-
	4×4	5.97E+05	1.78
	8×8	1.33E+06	2.23
	16×16	3.79E+06	2.84

Table 9.9: Stokes VGP with the scaled graph norm using Lobatto basis functions for the test space, Gram matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

9.5 Static Condensation: Effect on Stiffness Matrix Conditioning

The principal reason for employing static condensation (described in Chapter 2) is the significant reduction[1] in global degrees of freedom it offers us. We have also found, anecdotally, that it can improve the *effective* conditioning of the solve—i.e. in some instances, we get more accurate results when we use static condensation. One instance of this is discussed in Chapter 6; by using static condensation, we were able to resolve the fourth Moffatt eddy in the lid-driven cavity flow problem—without it, we could not.

It is therefore worth examining, in the context of the present chapter, what effect static condensation has on the condition number of the global

[1]For example, for a 16×16 quartic solution using the scaled graph norm—and therefore with quintic velocity field variables—the static condensation solve employs 16,771 global degrees of freedom, compared with 60,803 for the standard solve.

Order	Mesh Size	Condition #	Ratio
1	2×2	2.82E+03	-
	4×4	5.98E+03	2.12
	8×8	3.79E+04	6.33
	16×16	2.85E+05	7.53
2	2×2	3.88E+03	-
	4×4	1.12E+04	2.88
	8×8	7.23E+04	6.47
3	2×2	4.15E+04	-
	4×4	8.72E+04	2.10
	8×8	1.79E+05	2.05
4	2×2	6.58E+04	-
	4×4	1.38E+05	2.09
	8×8	3.34E+05	2.43

Table 9.10: Stokes VGP with the scaled graph norm using Lobatto basis functions for the test space, global stiffness matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

stiffness matrix (note that static condensation does not affect the Gram matrix). We continue with the example of the previous section, using the Stokes scaled graph norm and the Lobatto basis functions. The stiffness matrix condition numbers are listed in Table 9.11. Here, it appears that static condensation does not have a great effect in either direction on the condition numbers: in some cases there is modest degradation; in others, modest improvement. Our hypothesis is that static condensation improves the conditioning of the solve in ways that are not adequately reflected by the condition number of the matrix.

Order	Mesh Size	Condition #	Ratio
$k = 1$	2×2	3.03E+03	-
	4×4	8.68E+03	2.87
	8×8	4.58E+04	5.28
	16×16	3.36E+05	7.32
$k = 2$	2×2	6.89E+03	-
	4×4	1.68E+04	2.43
	8×8	8.19E+04	4.89
	16×16	5.95E+05	7.26
$k = 3$	2×2	1.09E+04	-
	4×4	2.66E+04	2.45
	8×8	1.42E+05	5.33
	16×16	1.04E+06	7.35
$k = 4$	2×2	1.56E+04	-
	4×4	3.87E+04	2.47
	8×8	2.27E+05	5.86
	16×16	1.68E+06	7.42

Table 9.11: Stokes VGP with the scaled graph norm, Lobatto basis functions for the test space, statically condensed global stiffness matrix condition numbers. A ratio of 4 corresponds to a $O(h^{-2})$ growth rate for the condition number.

9.6 Some Tentative Conclusions

In our basis conditioning studies, while we did not observe clear benefits in the condition numbers of the H^1 and $H(\text{div})$ mass and stiffness matrices for the Lobatto bases (with the exception of the $H(\text{div})$ stiffness matrix, which by construction exhibits perfect conditioning), we did observe improved condition numbers in both the Gram and stiffness matrices when we used the Lobatto bases for our test space discretization.

The studies of variations of the Stokes graph norm suggest a "short blanket" effect in the conditioning of the local and global solves: by employing h-scaling to improve the conditioning of the local solves, one is liable to worsen the conditioning of the global solve. Curiously, the graph norm scaled according to units in the dimensional equations strikes a balance between the conditioning of the global and local solves, such that each of these scales as $O(h^{-2})$.

However, it is worth mentioning that condition numbers are not the whole story—good condition numbers are sufficient to allow efficient, accurate solution of a system, but they may not be necessary. We have observed this in practice: we have seen Gram matrices with condition numbers as high as 10^{16}—which, according to the rule of thumb above, could mean a catastrophic loss of precision—in DPG computations that exhibited no discernible ill effects. Similarly, while we have some anecdotal evidence that static condensation improves the accuracy of our solves, in our tests static condensation sometimes resulted in larger condition numbers. A good area for further study would be

a more detailed examination of the spectrum of these matrices, which may tell us more about the solvability of these systems.

Chapter 10

Conclusions and Future Work

10.1 Summary Results

In this dissertation, we have applied DPG to a variety of model problems in steady 2D incompressible flow. Along the way, we have developed several DPG formulations for the Stokes equations—the VGP, VVP, and VSP formulations—as well as DPG formulations for the Oseen and linearized Navier-Stokes equations. We have proved mathematically and illustrated numerically that optimal convergence rates are achieved for Stokes when using the VGP formulation with the graph norm. We similarly achieved optimal convergence rates in numerical experiments for the other Stokes formulations, as well as the Oseen and Navier-Stokes formulations.

We further demonstrated the effectiveness of DPG's energy error measurement for robust adaptivity in the context of a variety of flow problems: the lid-driven cavity flow problem, the backward-facing step problem, and the flow past a cylinder problem.

We developed an alternative, scaled version of the graph norm for the VGP Stokes formulation that allows improved convergence rates for the velocity at minimal additional cost, and which also appears to offer better

conditioning properties, striking a balance between the conditioning of the global stiffness matrix and the local Gram matrices used in optimal test function determination.

Underpinning all of these efforts was Camellia, the software framework for rapid development of DPG solvers which we developed for this dissertation—offering convenient, modular interfaces for the definition of DPG bilinear forms and test space norms, providing simple, extensible mechanisms for h-, p-, and hp-adaptivity, and taking advantage of the embarrassingly parallel nature of optimal test function determination to provide scalable computation of the stiffness matrix, among other features.

These results demonstrate DPG's great promise in the context of incompressible flow problems: the fact that the method is stable even on extremely coarse meshes combined with the built-in error measure means that we may begin with a mesh that merely captures the geometry and perform automatic refinements to converge to a solution with desired accuracy. While the present numerical experiments are limited to two-dimensional, steady-state problems, both our analysis and the numerical results thus far warrant optimism for DPG applied to three-dimensional and transient problems. If fully realized, the capabilities of DPG we have explored here could eliminate the need to design complex grids to resolve expected features of the solution.

10.2 Future Research Directions

The DPG and Camellia research efforts thus far suggest several fruitful areas for future exploration.

10.2.1 Extending the Navier-Stokes Experiments

As noted at several points in Chapter 8, the emphasis in our numerical experiments in the present work has been on DPG's ability to adaptively converge to a solution that agrees qualitatively with previously reported results. With a few exceptions, we have not compared quantities of interest—such as drag coefficients and the dimensions of recirculation regions—with those previously reported. We plan to perform such comparisons in the future, as well as to examine the sensitivity of our results to the dimensions of the computational domain; in the presence of artificial boundary conditions such as those we impose on the outflow in both the backward-facing step and the flow past a cylinder problems, a domain of insufficient length can introduce errors in the solution (these arise from the disagreement of the conditions imposed with physical conditions at the outflow).

10.2.2 DPG and HPC

DPG has several features that make it a good candidate for future high-performance computing (HPC) applications. First, it is a high-order method; high-order methods are known to provide better computational *intensity*—a measure of the amount of computational effort expended relative to the

amount of communication required. Since the progression in HPC is toward higher relative communications costs—"flops will be free" has become a tagline in discussions of exascale computing—higher computational intensity is a desirable feature. DPG's optimal test function computation, being an element-local operation, similarly adds to the computational intensity—DPG's optimal test functions allow one to trade local effort for improved accuracy. Camellia's use of static condensation to reduce the global system to traces and flux coefficients also adds to the computational intensity: no information about field coefficients needs to be communicated between computational nodes.

The robust adaptivity offered by DPG also lends itself to HPC applications, in that it affords a level of *automaticity* uncommon in fluid dynamics computations—and, as discussed in Chapter 1, in typical applications, the Navier-Stokes solve is just one component in an optimization loop, which makes failures that require human intervention costly. HPC increases the speed at which computations can in principle be completed, magnifying the cost of any events that halt the computation.

One of the key areas that must be addressed for DPG to be applied to many HPC-scale problems is the development of a DPG solver that scales to thousands—if not tens of thousands, or hundreds of thousands—of processors. As mentioned above, Camellia already provides a highly scalable mechanism for computing the optimal test functions and the global stiffness matrix; however, how to solve the global matrix system in a robust, efficient way remains an open question for most problems—a notable exception is the

Poisson problem, for which Barker et al. have demonstrated that additive Schwarz preconditioners are effective [5].

There are two clear lines of research which one might pursue in this context. First, one might build on the work of Barker et al. to develop a general framework for preconditioning the DPG stiffness matrix—or absent that, one might at least develop good preconditioners for other problems. As mentioned in Chapter 1, another recent, and very promising, idea has recently been proposed by Bui-Thanh and Ghattas [20], who have developed a PDE-constrained optimization approach to DPG, providing a unified method for linear and nonlinear problems—and an iterative solver for DPG. It may very well be that this will hold the key to highly scalable DPG applications. We hope to add an implementation of their method to Camellia in the near future.

10.2.3 Extending Camellia to More Dimensions: 3D and Time

Several of our studies in this dissertation have been regime-limited in the sense that at a critical Reynolds number a flow becomes three-dimensional or transient. Furthermore, certain fluid phenomena—notably vortex stretching—only arise in three-dimensional flows. Therefore, we would like to add support for 3D elements of various topological types—tetrahedra, hexahedra, and pyramids, perhaps—to Camellia.

We would also like to provide some mechanism for transient solves. We plan to add support for *space-time elements*,[1] which compute on a time

[1]In 2011, Chan et al. solved a transient 1D convection-dominated diffusion problem

slab—a tensor product structure, extruding the spatial mesh in the time dimension. The time slab may be refined in both time and space dimensions. Solution coefficients are only stored for the current time slab—a relatively modest additional cost, given that by this mechanism one can bring the entire DPG apparatus to bear on the time dimension. Our immediate plan in this regard is to add support for space-time elements to Camellia, by adding a mechanism for temporal extrusion of arbitrary element discretizations—thereby allowing transient computations for all element types supported—now and in the future—by Camellia.

Looking further forward, we would like to explore applying the notion of *parareal-in-time* elements [63] to DPG, allowing parallel computation in the time domain as well, through a multi-scale technology. Such distribution of computations in the time domain will likely be necessary as we move toward exascale computing.

10.2.4 Variations on the DPG Theme

We would like to add support for continuous finite element spaces and a Python interface to Camellia, to enhance the breadth of Camellia's applicability and the ease with which end users may take advantage of it. This will allow both easier comparison with existing finite element technologies, as well as exploration of new variations on the DPG theme—the usual DPG methodology requires a discontinuous test space, for example, but there is no

using DPG with space-time elements [24].

essential requirement that the trial space be discontinuous. It is even possible to pursue a DPG-like scheme that involves continuous test functions—such an approach has recently been investigated by Dahmen et al. [33], as well as Broersen and Stevenson [17]. Having continuous finite elements available within Camellia would also facilitate certain forms of post-processing, notably the streamfunction solves in this dissertation (which we simply solved using DPG, which was not the most appropriate choice).

10.3 Conclusion

In this dissertation, we have demonstrated the applicability of DPG to various problems in incompressible fluid flow. We have thereby begun to deliver on DPG's great promise of accurate solutions, automatic stability, and robust adaptivity. We look forward to continuing this work, applying DPG and related methodologies to larger-scale problems and problems in three-dimensional and transient regimes.

Appendices

Appendix A

Distributing the Stiffness Matrix in Camellia

This appendix describes our approach to computing the global stiffness matrix in parallel in Camellia. It is worth noting that the attendant timing experiments were performed prior to the implementation of static condensation during the global solve; now that this has been implemented, we would expect the overall runtime to be reduced and the scalability to improve.

A.1 Extending Camellia for Distributed Stiffness Matrix Determination

Here, we describe the changes that we implemented within Camellia to allow the stiffness matrix determination to be executed efficiently within an MPI environment.

Camellia was originally written to run in serial, so a number of changes were required to allow MPI execution—the most significant of these had to do with element partitioning and partitioning the degree-of-freedom coefficients. Trilinos's Intrepid routines are designed to execute on batches of cells that are alike in topology and polynomial order. We allow meshes of non-uniform topology and polynomial order; thus we had implemented an ElementType

class which allowed us to identify elements that could be batched. Our Mesh class maintained data structures such that information pertaining to a particular ElementType could be accessed quickly—for example, Mesh stored a data structure that contained the vertex coordinates belonging to elements of a given ElementType.

The upshot of this design is that mesh information is already, in a sense, partitioned. However, there is no reason to expect that this will be a good partitioning for dividing up the work of stiffness matrix computation across MPI nodes—for this, we want our mesh partitioning to keep spatially adjacent elements together within a partition to minimize the amount of data that needs to be communicated during global stiffness matrix assembly.

Thus, to maintain both of our desired design features—batching by ElementType and spatially contiguous partitioning across MPI nodes—we require a two-level partitioning; we first divide the elements spatially (the MPI partitions), and then batch together elements of like ElementType within each of these partitions. (Unfortunately, this is not quite the whole story; because other classes depend on the ElementType division of the whole mesh, we have had to maintain the old lookup tables as well. We do hope to eliminate these dependencies in the end.)

To facilitate experiments with multiple partitioning strategies (and to maintain separation of concerns), we designed an abstract MeshPartitionPolicy class which, given a Mesh and the number of partitions desired, returns a partition of the elements in the mesh. The Mesh then uses this to generate a

partition of the degree-of-freedom coefficients, which induces a partition of the rows in the stiffness matrix. The latter partition is used in conjunction with an Epetra FECrsMatrix, a distributed stiffness matrix class within Trilinos. The FECrsMatrix allows storage of data not owned by the current MPI node; this data is communicated to the owning MPI node during global assembly of the stiffness matrix. A convenient consequence of this is that we can separate the cost of global assembly from the cost of local stiffness matrix computations; the latter are independent across MPI nodes, and therefore we expect to see perfect scaling for these. The scaling of the global assembly will depend on the quality of the partition and the cost of communication between MPI nodes. We do not hope for perfect scaling here, but we do hope to see a cost per degree of freedom per node that does not grow too quickly in the total number of nodes.

The MeshPartitionPolicy partitions the mesh into sets of elements assigned to an MPI node; for the interface to Epetra, we need a partition of rows of the global stiffness matrix (each of which corresponds to a degree of freedom). Degrees of freedom corresponding to numerical fluxes are shared by the elements along whose boundaries they lie. We adopt a simple greedy policy for assignment of these d.o.f.s: shared d.o.f.s are assigned to the element with the smallest global identifier. This means that even if the element partitioning were perfectly balanced, some MPI nodes might still have more d.o.f.s assigned to them than others. This will not affect the amount of work involved in local stiffness matrix computations, but it may affect MPI communication costs

during global assembly.

Design Limitations: where Amdahl's law might come to haunt us. Here, we note a few features of the present design that may become bottlenecks as we scale up to larger problems and larger numbers of MPI nodes.

At present, our global solve is implemented using the KLU implementation provided by the Amesos package within Trilinos. This is a direct solve, executed in serial. Amesos gathers the global stiffness matrix to rank 0, solves, and then scatters the solution according to the partition we have defined. For our serial computations prior to the present work, the cost of the global solve has been dominated by the cost of the local optimal test function determination; however, now that the latter cost has been distributed using MPI and the former has not, we expect the global solve to become a bottleneck, both in terms of overall execution time and in terms of the sizes of problems that can be solved, since the KLU solve must fit into a single node's memory to be practical.

Similarly, at present the entire mesh is determined and stored on every MPI node. This has the advantage that no MPI communication is required during mesh construction, but for very large meshes it may become impractical, because the time cost of mesh construction may grow relative to the distributed parts of the computation, and perhaps (for *extremely* large meshes) because the memory required for the mesh may grow too large for storage on a node.

Finally, certain features of the original code depend upon all the solution coefficients being available to the current processor, so for the time being we distribute the entire solution to every MPI node, after Amesos has scattered according to the original partitioning. We hope to eliminate this dependence on having all solution coefficients available, allowing the solution to be distributed according to the same mapping as was used for the stiffness matrix.

A.1.1 Partitioning

Since Camellia is tightly integrated with the Trilinos library already, we use the Zoltan package in Trilinos to generate spatially local partitions of the mesh [42]. We interface specifically with the Hilbert Space-filling Curve and Reftree partitioning algorithms in Zoltan, both of which can include parallel implementations.

The Hilbert space-filling curve (HSFC) algorithm works by drawing a curve through the geometric mesh in 2- or 3-dimensional space, then mapping that curve to the interval $[0, 1]$. The partitioner then divides up this interval (in an even or weighted manner) into n partitions, and uses this partitioning to induce a partition of the mesh. All that's required by the HSFC algorithm is a listing of elements and a geometric identifier associated with a given element. In this case, we use the element centroid as such an identifier.

The Reftree algorithm [65] adds complexity in that each processor must store information not only about the active nodes, but the ancestors as well.

Reftree also requires that if you store an ancestor, you also store all its children as well, increasing the amount of redundancy between distributed refinement trees. However, since Reftree is not tuned to a particular element types (quad or triangle, tetrahedron or cube), the same algorithm performs similarly for many different types of refined meshes (as opposed to HSFC, which works best for quadrilateral grids, but not for triangular grids). Additionally, Reftree's algorithm guarantees connected partitions for tetrahedral and triangular grids, and generally performs more robustly on other types of grids as well.

Currently, we have implemented and tested a Reftree MeshPartitionPolicy on our workstations. However, Reftree currently crashes during runs on Lonestar, our HPC architecture for this report. We used the HSFC partitioner for the scaling tests in this report, and hope to fix issues with Reftree in the near future.

A.1.2 Scaling tests

We tested the scalability of our code on the Lonestar machine at the Texas Advanced Computing Center (TACC), verifying whether or not we observe strong/weak scaling for various parts of the program. We solve the convection-diffusion equation on a square mesh, with boundary conditions such that the solution develops a boundary layer (sharp gradient) near the north and east edges of the square. The solution for a refined mesh is shown in Figure A.1.

We set up our computational experiments by first refining all cells that

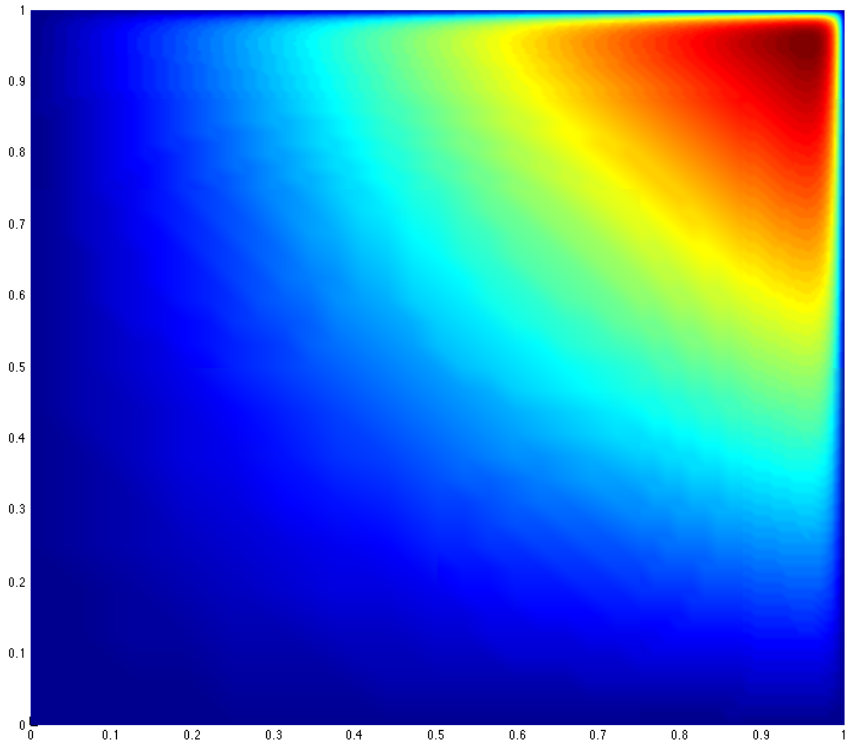


Figure A.1: Computed solution for $\epsilon = 10^{-2}$ using a 12,928-element mesh. L^2 error is $O(10^{-5})$.

share an edge with the north and east sides, and repeating this process four times. We then constructed three additional meshes, by uniformly refining every element in the mesh (quadrupling the number of elements each time). For each of these meshes, we solved on Lonestar, using 1, 4, 16, and 64 nodes, tracking the timing of various portions of the code. From this data, we extracted strong and weak scaling statistics, discussed below in Sections A.1.3

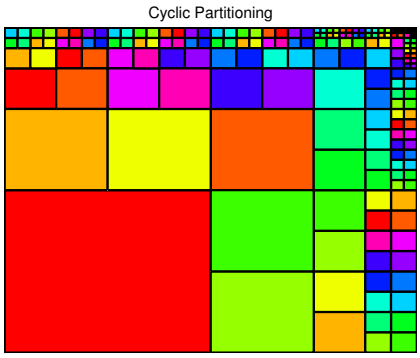
and A.1.4. The runtimes plotted in each of our figures are the mean of the times recorded at each MPI node.

We also used contiguous and discontiguous partitions of each mesh to measure the effect of partition quality on the communication costs in global stiffness matrix assembly. We created contiguous partitions of the mesh using a HSFC partitioner. To create the discontiguous partitioning, we used the Zoltan cyclic partitioner, which loops through the list of active elements, sending the first element to the first processor, the next to the next processor, and so on. After reaching the last processor, the cyclic partitioner cycles back to the first processor and repeats this process.[1] The cyclic partitioning is a "worst case" scenario in the sense that each of the current MPI node's elements' neighbors are owned by some other MPI node, with the consequence that information concerning each element boundary must be sent via MPI. An example cyclic partitioning can be seen in Figure A.2(a); Figure A.2(b) shows an HSFC partitioning.

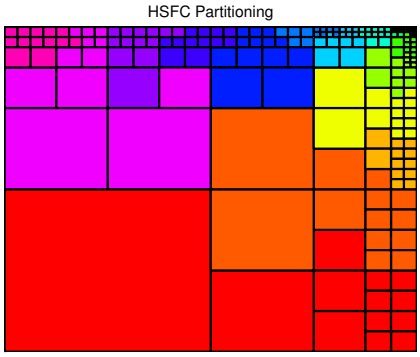
It's worth noting that the runtimes reported here do not represent Camellia running at peak efficiency. A single Lonestar node contains 12 cores. Because MPI communication has much higher bandwidth intra-node than inter-node, and for analysis we wanted approximately equal bandwidth

[1]Due to the nature of our refinements in this test, a blocked partitioner (equally partitioning the ordered list of active elements by element number) happened to produce nearly-contiguous partitions. We therefore have omitted this partitioner from our analysis, since we do not believe that its performance here accurately predicts its performance in general.

between MPI processes, we use only one core per node. Utilizing all 12 cores on a node would decrease communication costs. Additionally, there is a parameter within Camellia that controls the batch size for the Trilinos Intrepid batching routines. Tuning this parameter based on machine-specific cache sizes should further improve performance.



(a) Cyclic partitioning



(b) HSFC partitioning

Figure A.2: Discontiguous and contiguous partitions of the mesh.

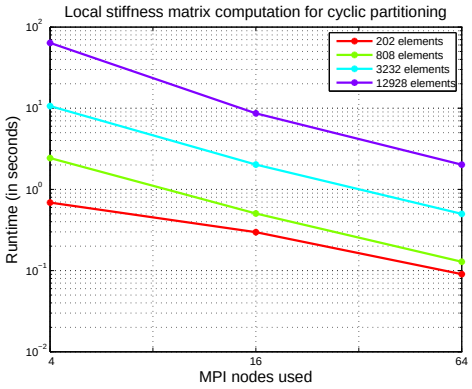
A.1.3 Strong scaling

In our strong scaling tests, we considered problems of fixed size, distributing the workload for each among increasing numbers of MPI nodes. Here, we hope to see runtimes that decrease linearly in the number of MPI nodes—this is what we mean by achieving strong scalability.

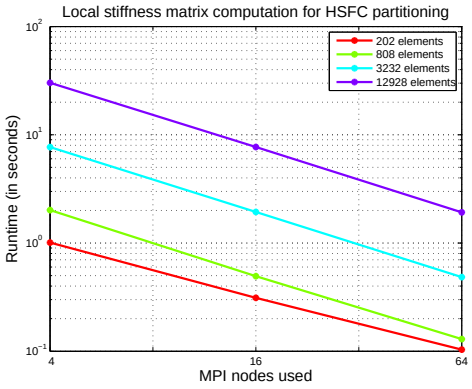
Because the local stiffness computations are entirely independent of each other, requiring no communication among MPI nodes, we expect to see perfect scaling for these. As can be seen in Figures A.3(a) and A.3(b), we do achieve something very close to this, although the cyclic partitioning took longer in almost every case, for reasons that are unclear to us. It may have something to do with data locality on each node—by the manner of our construction, data concerning contiguous mesh elements is more likely to be close together in memory than the data for mesh elements in the cyclic case.

It is less clear what we should expect for the global stiffness matrix assembly; we expect the HSFC partitioning to outperform the cyclic partitioning, and we would like to see the runtimes for assembly decrease as the number of MPI nodes increases. As can be seen in Figures A.4(a) and A.4(b), for the larger problems in both cases, we do have something approaching strong scalability. For the 202- and 808-element problems, we see the time cost increase (or fail to decrease, in the case of the 808-element problem for the cyclic partitioner) between 16 and 64 nodes. However, it's worth noting that in these cases, the real times involved are small: less than a second in each case. If our concern is our ability to scale up to large problems on large

numbers of processors, it does not appear that the global assembly will be a bottleneck. The HSFC partitioning does show its strength here compared with the cyclic; for the 12,928 mesh on the 4-node run, the cyclic partitioning took an average of 49.1 seconds per node, compared with 1.63 seconds for the HSFC partitioning.

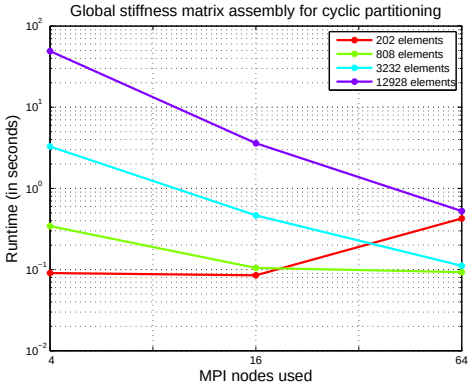


(a) Cyclic Partitioning

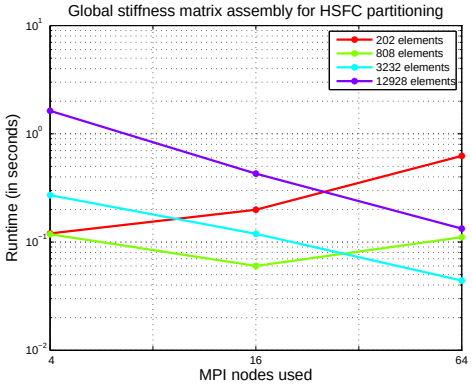


(b) HSFC Partitioning

Figure A.3: Strong scaling plots for the local stiffness matrix computation.



(a) Cyclic Partitioning

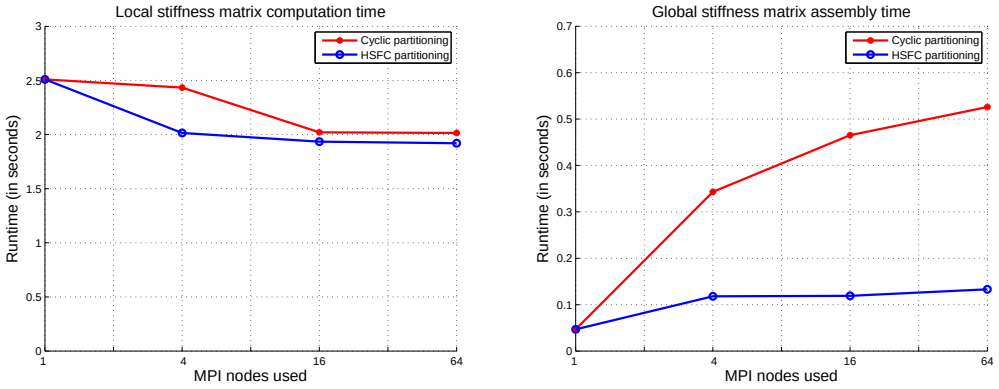


(b) HSFC Partitioning

Figure A.4: Strong scaling plots for global stiffness matrix assembly. (Note the differing scales.)

A.1.4 Weak scaling

In our weak scaling tests, we aimed to fix the size of the problem *per MPI node*, and examined the runtime of various portions of the execution—we hope to see a constant runtime as the problem size and number of processors increase, and this is what we mean by achieving weak scalability.



(a) Local stiffness computation (b) Global stiffness assembly

Figure A.5: Weak scaling results for computation of local stiffness matrices and global stiffness matrix assembly. For each run, both the number of MPI nodes and the number of elements increased by a factor of four.

Both partition types achieve weak scalability in the local stiffness matrix computation, as can be seen in Figure A.5(a). In fact, we see the total runtime *decrease* slightly as the total workload and number of MPI nodes increase—we are unsure of the reason for this. Although we would not expect the partitioning to affect this portion of the computation at all (since no inter-node communication occurs at this stage), for some reason the HSFC partitioning slightly outperforms the cyclic on each of the multi-node runs. As with the strong scaling, we speculate that this is due to locality of the owned element data in the case of the HSFC partitioning.

Assembly of the global stiffness matrix achieves weak scalability only for the contiguous case, as can be seen in Figure A.5(b). It's worth noting, however, that though the poor mesh partitioning imposes much higher

communication costs than the HSFC partitioning, the maximum amount of time spent on assembly is still dominated by the time spent on computation of local stiffness matrices.

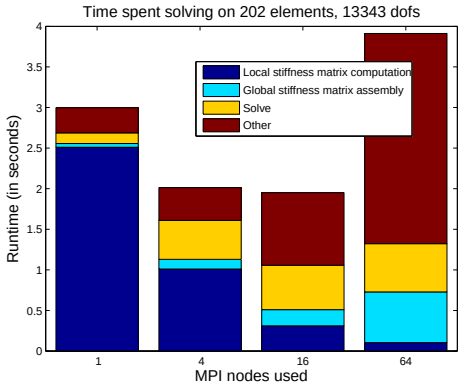
A.1.5 Total runtime analysis

As the final component of our performance analysis, we examine components of the total runtime for each of our four problems using 1, 4, 16, and 64 MPI nodes; here we use the HSFC partitioner. This can be seen in Figure A.6. We see that for the largest MPI runs the combined time spent on local stiffness computation and global assembly is a small fraction of the total time spent—since these were the targets of the present work, we can count it a success. The graphs also show that the cost of the global solve dominates in each of these runs, making this the obvious next target for optimization. In the "Other" category, the largest cost is the mesh determination. In the 12,928-element case, this takes about 5 seconds. Thus, a distributed mesh might be our next target after the solve is optimized.

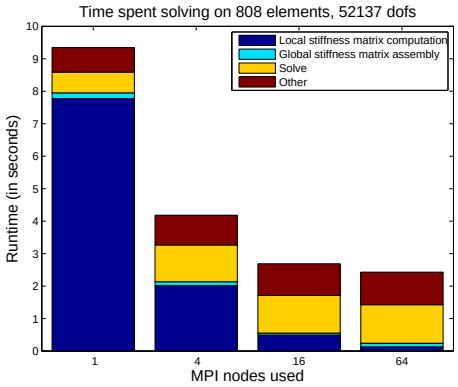
A.1.6 Future Work for Better Scaling

There are several ways in which we hope to extend Camellia further to allow it to solve larger problems still; we hope to:

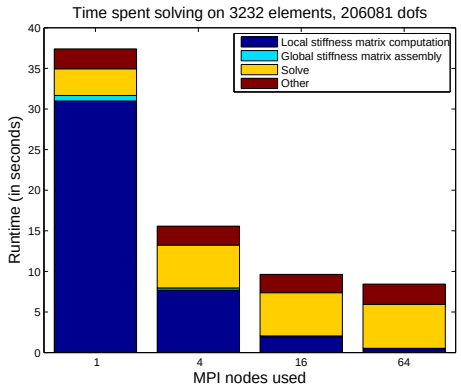
- distribute the solve, using MUMPS or an iterative solver,
- improve the load balancing for meshes of variable polynomial order,



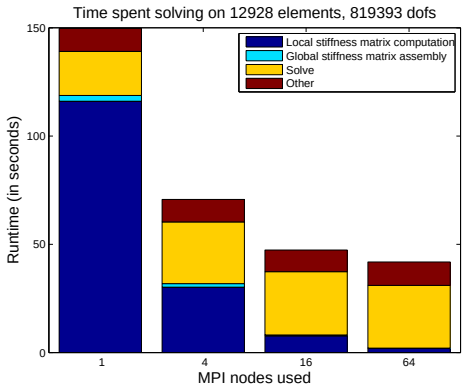
(a) 202 elements, 13,343 dofs



(b) 808 elements, 52,137 dofs



(c) 3,232 elements, 206,081 dofs



(d) 12,928 elements, 819,393 dofs

Figure A.6: Breakdown of time spent in a solve for a series of HSFC-partitioned meshes.

- distribute mesh construction and storage, and
- distribute solution storage.

As discussed in Section A.1.5 above, the lowest-hanging fruit for further improving the performance for the experimental setup considered in the

present work is distributing the solve. We have already successfully interfaced with MUMPS—a parallel sparse direct solver—on one of our laptops; we expect that getting MUMPS to work on Lonestar will just be a matter of building Trilinos and Camellia with MUMPS there. One of the advantages of DPG is that its stiffness matrices are guaranteed to be symmetric positive definite, which opens the door to using iterative methods such as conjugate gradient methods. DPG also produces matrices whose entries are coupled only through the flux degrees of freedom; thus static condensation techniques can be used to reduce the size of the global system to be solved. We believe that iterative solvers and static condensation would allow us to solve larger problems than MUMPS will allow—we have heard that MUMPS only scales to about 128 processors.

In the present work, we considered a mesh of uniform polynomial order and considered all elements to be of equal weight; Camellia supports meshes with variable order, and Zoltan provides facilities for weighting elements differently. By taking advantage of this feature of Zoltan (using a weight corresponding to the number of degrees of freedom on each element), we can improve Camellia's load balancing for meshes of variable polynomial order. In fact, in doing so, we expect to improve slightly the load balancing for the setup considered here, because elements with hanging nodes do have slightly more degrees of freedom than those without.

At present, the mesh is constructed and stored on each node, as is the solution. It should be a relatively small extension to the present work to allow

the solution to be distributed. Allowing mesh construction to be distributed is a bigger challenge, largely because the nodes must agree on the global stiffness matrix row indices. Given that we are working with an adaptive mesh, it will likely make sense to distribute the work of mesh refinement according to the partition of the previous mesh and use Zoltan to do the load rebalancing.

In summary, we have achieved good overall speedup for Camellia using a Hilbert space-filling curve partitioner, and have identified the next steps for improving its parallel performance further.

Appendix B

Further Code Verification for Camellia: Poisson and Stokes VSP, VVP Formulations

In this appendix, we include the results of several of our verification tests for Camellia in the context of a DPG formulation for the Poisson problem, as well as the VVP and VSP Stokes formulations defined in Chapter 2. The DPG formulation of the Poisson problem, which we use is given by

$$\begin{aligned} b(\cdot,\cdot) = &- \int_K \phi \nabla \cdot \mathbf{q} - \int_{\mathbf{K}} \psi \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n} \\ &- \int_K \psi \cdot \nabla v + \int_{\partial K} \widehat{\psi}_n \, v \,. \end{aligned}$$

For each of our three formulations — Poisson, VSP Stokes, and VVP Stokes — we have run a set of numerical experiments:

- For L^2 polynomial orders 1, 2, and 3, run convergence studies on meshes varying from 1×1 to 32×32.
- For a 16×16 mesh, vary the polynomial orders of the elements across the mesh.
- For L^2 polynomial orders 1, 2, and 3, run convergence studies on "hybrid" meshes (quads and triangles together) varying in size from 1×1 to 32×32.

The first two experiments we ran with triangles and quads. For all of the experiments, we specified the domain $(-1,1) \times (-1,1)$, imposed a zero-mean constraint, and used the naive test space norm. As will be clear from the discussion in Chapters 4 and 6, the naive norm is not an ideal norm for our Stokes formulations, as it leads to suboptimal convergence in the pressure; we performed these tests prior to performing that analysis, and in the future we hope to repeat these tests with the graph norms corresponding to the VSP and VVP formulations. We used the Poisson manufactured solution

$$\phi = e^{x \sin y} - \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} e^{x \sin y} dx dy,$$

where the subtracted integral is chosen to ensure that ϕ does indeed have a zero mean. Following our previous work, the Stokes experiments used a manufactured solution

$$\begin{aligned} u_1 &= -e^x (y \cos y + \sin y) \\ u_2 &= e^x y \sin y \\ p &= 2\mu e^x \sin y. \end{aligned}$$

Convergence Studies on Uniform Meshes For polynomial order k, we expect a convergence rate of $k+1$. This is indeed what we see in our experiments. The Poisson convergence studies with triangular elements can be found in Table B.1; with quads, in Table B.2. The Stokes VSP studies with triangles can be found in Table B.3; with quads, in Table B.4. The VVP studies with triangles are shown in Table B.5; with quads, in Table B.6. It is

worth noting that the convergence rates we see with the naive test space norm for the VSP and VVP are better than the rates we saw when using the naive norm with the VGP formulation (see Chapter 6); however, the L^2 errors in pressure remain significantly larger than the best approximation error.

Meshes of Multiple Polynomial Orders To confirm that our code works well with meshes that include elements of varying degree, we took a 16×16 mesh with L^2 polynomial degree assigned according to the following pattern, repeated 4 times:

4	4	4	4	1	1	1	1	2	2	2	2	3	3	3	3
3	3	3	3	4	4	4	4	1	1	1	1	2	2	2	2
2	2	2	2	3	3	3	3	4	4	4	4	1	1	1	1
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4

We hope to see errors as good as, or better than, our first-order 16×16 mesh for the same problem, although we can perhaps explain a small amount of increased error as due to worse conditioning of the matrices for higher-order polynomials.

The results for Poisson are:

Triangles					
ϕ		ψ_1		ψ_2	
$k=1$	mixed k	$k=1$	mixed k	$k=1$	mixed k
2.0e-3	9.1e-4	3.4e-3	1.7e-3	2.4e-3	1.1e-3
Quads					
ϕ		ψ_1		ψ_2	
$k=1$	mixed k	$k=1$	mixed k	$k=1$	mixed k
1.0e-3	3.7e-4	2.3e-3	6.6e-4	2.9e-3	1.2e-3

For Poisson, the multi-order mesh has lower error than the first-order mesh in every variable, just as we would like.

The results for Stokes VSP are:

Triangles					
p		u_1		u_2	
$k=1$	mixed k	$k=1$	mixed k	$k=1$	mixed k
1.6e-2	1.4e-2	5.1e-3	2.4e-3	4.4e-3	2.1e-3
Quads					
p		u_1		u_2	
$k=1$	mixed k	$k=1$	mixed k	$k=1$	mixed k
5.3e-3	1.0e-2	4.9e-3	1.6e-3	2.5e-3	1.3e-3

For Stokes VSP, the multi-order mesh has lower error than the first-order mesh in every variable, except for pressure in the quad mesh, where the first-order mesh does better by about a factor of 2.

The results for Stokes VVP are:

Triangles					
p		u_1		u_2	
$k=1$	mixed k	$k=1$	mixed k	$k=1$	mixed k
6.9e-2	5.8e-2	6.9e-3	3.1e-3	6.8e-3	2.7e-3
Quads					
p		u_1		u_2	
$k=1$	mixed k	$k=1$	mixed k	$k=1$	mixed k
8.0e-3	2.1e-2	3.9e-3	1.7e-3	3.8e-3	1.4e-3

Again, the behavior is just as we would like, except in the case of pressure for quads, where the error in the first-order mesh is better than that for the multi-order, this time by a factor of about 2.5. We believe that this is due to round-off error, but we do not at present have a precise explanation.

"Hybrid" Mesh Convergence Studies We studied the convergence of the Poisson solution for "hybrid" meshes, containing half triangles and half quads, ranging from 1×1 to 32×32, with L^2 polynomial orders $k = 1, 2, 3$. The results can be seen in Table B.7; again, the convergence rates for all variables are optimal. Plots of ϕ, ψ_1 and ψ_2 for the 16×16 mesh can be found in Figure B.1.

We studied the same for the VSP and VVP Stokes formulations. VSP results can be found in Table B.8; VVP, Table B.9 — the rates are optimal for u_1 and u_2 across the board, but for reasons unknown the $k = 1$ VVP pressure is somewhat sub-optimal, while for $k = 2$ and 3, the rate is somewhat super-optimal. Something similar happens in the VSP pressure, although there the effect is considerably less pronounced. Plots of P, u_1 and u_2 for the

16×16 mesh can be found in Figure B.2. These were generated using the VVP formulation; the VSP plots are visually identical.

B.1 Results of Convergence Studies on Uniform Meshes

k=1						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	4.2e-1	-	5.2e-1	-	3.4e-1	-
2×2	1.3e-1	1.71	2.0e-1	1.38	1.5e-1	1.24
4×4	3.2e-2	2.00	5.2e-2	1.93	3.9e-2	1.92
8×8	8.0e-3	2.01	1.3e-2	1.96	9.6e-3	2.00
16×16	2.0e-3	2.00	3.4e-3	1.99	2.4e-3	2.01
32×32	5.0e-4	2.00	8.4e-4	1.99	6.0e-4	2.00
k=2						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	7.9e-2	-	2.5e-1	-	1.4e-1	-
2×2	1.2e-2	2.77	3.1e-2	2.97	3.6e-2	1.96
4×4	1.4e-3	3.00	4.1e-3	2.95	4.9e-3	2.86
8×8	1.8e-4	2.99	5.1e-4	2.98	6.0e-4	3.02
16×16	2.3e-5	3.00	6.5e-5	2.99	7.5e-5	3.01
32×32	2.8e-6	3.00	8.1e-6	3.00	9.3e-6	3.01
k=3						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	2.5e-2	-	2.9e-2	-	4.3e-2	-
2×2	1.5e-3	3.99	3.4e-3	3.09	5.2e-3	3.06
4×4	1.1e-4	3.82	2.3e-4	3.92	3.4e-4	3.95
8×8	7.1e-6	3.94	1.5e-5	3.99	2.1e-5	3.98
16×16	4.5e-7	3.99	9.2e-7	3.99	1.3e-6	4.00
32×32	2.8e-8	4.00	5.8e-8	4.00	8.4e-8	4.00

Table B.1: Poisson: Triangles, L^2 Error and h-Convergence Rates. We observe optimal convergence.

k=1						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	1.4e-1	-	5.4e-1	-	3.8e-1	-
2×2	5.2e-2	1.41	1.3e-1	2.00	1.4e-1	1.41
4×4	1.5e-2	1.77	3.5e-2	1.92	4.3e-2	1.70
8×8	4.1e-3	1.93	9.1e-3	1.97	1.2e-2	1.91
16×16	1.0e-3	1.98	2.3e-3	2.00	2.9e-3	1.98
32×32	2.6e-4	2.00	5.7e-4	2.00	7.3e-4	2.00
k=2						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	4.9e-2	-	8.5e-2	-	1.1e-1	-
2×2	6.6e-3	2.91	1.7e-2	2.34	1.6e-2	2.78
4×4	7.8e-4	3.08	2.2e-3	2.90	1.8e-3	3.12
8×8	9.3e-5	3.05	2.6e-4	3.11	2.0e-4	3.20
16×16	1.2e-5	3.01	3.1e-5	3.06	2.3e-5	3.11
32×32	1.4e-6	3.00	3.8e-6	3.03	2.8e-6	3.05
k=3						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	1.2e-2	-	3.0e-2	-	2.6e-2	-
2×2	6.6e-4	4.16	2.6e-3	3.54	2.0e-3	3.66
4×4	3.3e-5	4.33	1.2e-4	4.39	1.1e-4	4.18
8×8	2.1e-6	3.99	7.3e-6	4.09	6.6e-6	4.10
16×16	1.3e-7	3.99	4.4e-7	4.04	3.9e-7	4.07
32×32	8.1e-9	4.00	2.7e-8	4.02	2.4e-8	4.04

Table B.2: Poisson: Quads, L^2 Error and h-Convergence Rates. We observe optimal convergence.

k=1						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	3.2e-0	-	9.4e-1	-	1.6e-0	-
2×2	1.1e-0	1.53	3.1e-1	1.62	2.8e-1	2.05
4×4	3.0e-1	1.89	8.0e-2	1.94	7.0e-2	1.99
8×8	6.9e-2	2.11	2.0e-2	1.99	1.8e-2	2.00
16×16	1.6e-2	2.12	5.1e-3	2.00	4.4e-3	2.00
32×32	3.8e-3	2.08	1.3e-3	2.00	1.1e-3	2.00
k=2						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	2.3e-0	-	2.6e-1	-	4.3e-1	-
2×2	2.8e-1	3.02	4.3e-2	2.63	5.1e-2	3.05
4×4	3.4e-2	3.06	5.8e-3	2.88	6.6e-3	2.97
8×8	4.0e-3	3.10	7.4e-4	2.97	8.3e-4	2.99
16×16	4.7e-4	3.06	9.3e-4	2.99	1.0e-4	3.00
32×32	5.8e-5	3.03	1.2e-5	3.00	1.3e-5	3.00
k=3						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	3.7e-1	-	5.0e-2	-	3.7e-2	-
2×2	3.4e-2	3.46	4.0e-3	3.66	3.4e-3	3.44
4×4	2.6e-3	3.70	2.6e-4	3.95	2.2e-4	3.94
8×8	1.9e-4	3.82	1.6e-5	3.99	1.4e-5	4.00
16×16	1.1e-5	4.06	1.0e-6	4.00	8.8e-7	4.00
32×32	5.8e-7	4.27	6.4e-8	4.00	5.5e-8	4.00

Table B.3: Stokes VSP: Triangles, L^2 Error and h-Convergence Rates. We observe optimal convergence rates for the velocity. The pressure rates are *super*-optimal, which demonstrates that these cannot be the asymptotic values. We believe the error in the pressure remains larger than the best approximation error due to our poor choice of test space norm (the naive norm).

k=1						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	1.1e-0	-	4.7e-1	-	8.3e-1	-
2×2	6.1e-1	0.84	3.0e-1	0.65	1.7e-1	2.33
4×4	1.4e-1	2.07	7.8e-2	1.94	4.0e-2	2.03
8×8	2.9e-2	2.33	2.0e-2	1.99	1.0e-2	2.01
16×16	5.4e-3	2.41	4.9e-3	2.00	2.5e-3	2.00
32×32	1.0e-3	2.38	1.2e-3	2.00	6.3e-4	2.00
k=2						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	3.3e-1	-	3.0e-1	-	1.7e-1	-
2×2	1.8e-1	0.82	3.2e-2	3.22	2.0e-2	3.06
4×4	1.5e-2	3.67	3.9e-3	3.04	2.5e-3	3.02
8×8	1.2e-3	3.58	4.8e-4	3.01	3.1e-4	3.01
16×16	1.0e-4	3.54	6.0e-5	3.00	3.9e-5	3.00
32×32	9.5e-6	3.46	7.6e-6	3.00	4.9e-6	3.00
k=3						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	2.7e-1	-	1.3e-2	-	3.3e-2	-
2×2	2.0e-2	3.74	1.7e-3	2.91	1.8e-3	4.17
4×4	1.3e-3	4.02	1.1e-4	3.96	1.1e-4	4.05
8×8	7.0e-5	4.16	6.8e-6	3.99	6.9e-6	4.02
16×16	3.6e-6	4.29	4.3e-7	4.00	4.3e-7	4.00
32×32	1.6e-7	4.52	2.7e-8	4.00	2.7e-8	4.00

Table B.4: Stokes VSP: Quads, L^2 Error and h-Convergence Rates. We observe optimal convergence rates for the velocity. The pressure rates are *super*-optimal, which demonstrates that these cannot be the asymptotic values. Comparison with the best approximation values determined in our VGP experiments verifies that the error in the pressure is considerably larger than the best approximation error in the discrete space (e.g. for $k = 3$ and a 16×16 mesh, the best approximation error is 6.8e-8, vs. 3.6e-6 here); the larger error in the pressure is almost certainly due to our use of the naive test space norm.

k=1						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	3.0e-0	-	1.6e-0	-	2.0e-0	-
2×2	1.5e-0	1.00	4.9e-1	1.72	4.2e-1	2.23
4×4	6.4e-1	1.23	1.1e-1	2.14	1.1e-1	1.90
8×8	2.3e-1	1.51	2.8e-2	2.00	2.8e-2	2.02
16×16	6.9e-2	1.70	6.9e-3	2.00	6.8e-3	2.02
32×32	2.0e-2	1.77	1.7e-3	2.00	1.7e-3	2.01
k=2						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	2.1e-0	-	3.3e-1	-	4.6e-1	-
2×2	8.5e-2	4.65	5.5e-2	2.58	5.6e-2	3.04
4×4	8.3e-3	3.35	7.3e-3	2.90	7.7e-3	2.85
8×8	7.7e-4	3.44	9.3e-4	2.97	1.0e-3	2.96
16×16	6.8e-5	3.51	1.2e-4	2.99	1.3e-4	2.99
32×32	6.9e-6	3.30	1.5e-5	3.00	1.6e-5	3.00
k=3						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	1.7e-1	-	5.9e-2	-	5.4e-2	-
2×2	1.3e-2	3.74	4.4e-3	3.74	4.2e-3	3.68
4×4	8.9e-4	3.83	2.9e-4	3.93	2.7e-4	3.96
8×8	4.8e-5	4.22	1.8e-5	3.99	1.7e-5	3.99
16×16	2.0e-6	4.60	1.1e-6	4.00	1.1e-6	4.00
32×32	6.7e-8	4.89	7.2e-8	4.00	6.4e-8	4.00

Table B.5: Stokes VVP: Triangles, L^2 Error and h-Convergence Rates. We observe optimal convergence rates for the velocity. The pressure rates are *super*-optimal, which demonstrates that these cannot be the asymptotic values. We believe the error in the pressure remains larger than the best approximation error due to our poor choice of test space norm (the naive norm).

k=1						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	1.5e-0	-	4.9e-1	-	1.5e-0	-
2×2	1.5e-0	0.02	3.6e-1	0.45	3.1e-1	2.26
4×4	1.0e-1	3.90	6.5e-2	2.45	6.3e-2	2.28
8×8	3.0e-2	1.74	1.6e-2	2.05	1.5e-2	2.04
16×16	8.0e-3	1.91	3.9e-3	2.01	3.8e-3	2.01
32×32	2.0e-3	1.97	9.8e-4	2.00	9.6e-4	2.00
k=2						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	2.1e-0	-	3.9e-1	-	2.7e-1	-
2×2	5.1e-2	5.33	3.3e-2	3.56	2.2e-2	3.60
4×4	3.7e-3	3.80	3.9e-3	3.07	2.6e-3	3.06
8×8	3.5e-4	3.41	4.9e-4	3.02	3.3e-4	3.01
16×16	3.8e-5	3.18	6.1e-5	3.00	4.1e-5	3.00
32×32	4.4e-6	3.12	7.6e-6	3.00	5.1e-6	3.00
k=3						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	1.9e-1	-	1.3e-2	-	3.7e-2	-
2×2	1.3e-2	3.82	1.8e-3	2.85	2.0e-3	4.23
4×4	4.7e-4	4.83	1.1e-4	3.97	1.1e-4	4.11
8×8	1.9e-5	4.59	7.1e-6	4.00	7.1e-6	4.02
16×16	1.0e-6	4.24	4.4e-7	4.00	4.4e-7	4.00
32×32	6.3e-8	4.04	2.8e-8	4.00	2.8e-8	4.00

Table B.6: Stokes VVP: Quads, L^2 Error and h-Convergence Rates. We observe optimal convergence rates for the velocity. The pressure rates are *super*-optimal, which demonstrates that these cannot be the asymptotic values. Comparison with the best approximation values determined in our VGP experiments verifies that the error in the pressure is considerably larger than the best approximation error in the discrete space (e.g. for $k = 3$ and a 16×16 mesh, the best approximation error is 6.8e-8, vs. 1.0e-6 here); the larger error in the pressure is almost certainly due to our use of the naive test space norm.

B.2 Results of Convergence Studies on Hybrid Meshes

k=1						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	4.2e-1	-	5.2e-1	-	3.4e-1	-
2×2	9.6e-2	2.12	1.7e-1	1.58	1.4e-1	1.27
4×4	2.4e-2	1.99	4.5e-2	1.93	4.0e-2	1.84
8×8	6.1e-3	2.00	1.2e-2	1.97	1.0e-2	1.96
16×16	1.5e-3	2.00	2.9e-3	1.99	2.6e-3	1.99
32×32	3.8e-4	2.00	7.3e-4	2.00	6.4e-4	2.00
k=2						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	7.9e-2	-	2.5e-1	-	1.4e-1	-
2×2	9.5e-3	3.06	2.5e-2	3.29	2.8e-2	2.33
4×4	1.2e-3	3.03	3.3e-3	2.94	3.7e-3	2.90
8×8	1.4e-4	3.01	4.1e-4	3.01	4.5e-4	3.04
16×16	1.8e-5	3.00	5.1e-5	3.01	5.5e-5	3.02
32×32	2.3e-6	3.00	6.3e-6	3.00	6.9e-6	3.01
k=3						
Mesh Size	ϕ	rate	ψ_1	rate	ψ_2	rate
1×1	2.5e-2	-	2.9e-2	-	4.3e-2	-
2×2	1.2e-3	4.37	3.0e-3	3.27	4.0e-3	3.45
4×4	8.1e-5	3.88	1.8e-4	4.05	2.5e-4	3.98
8×8	5.2e-6	3.95	1.2e-5	3.99	1.6e-5	3.99
16×16	3.3e-7	3.99	7.2e-7	4.00	9.9e-7	4.00
32×32	2.1e-8	4.00	4.4e-8	4.00	6.2e-8	4.00

Table B.7: Poisson: "Hybrid" Mesh, L^2 Error and h-Convergence Rates. We observe optimal convergence.

k=1						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	3.2e-0	-	9.4e-1	-	1.2e-0	-
2×2	1.1e-0	1.58	3.1e-1	1.62	2.7e-1	2.10
4×4	2.9e-1	1.88	8.0e-2	1.94	6.8e-2	1.99
8×8	6.7e-2	2.12	2.0e-2	1.99	1.7e-2	2.00
16×16	1.5e-2	2.14	5.0e-3	2.00	4.2e-3	2.00
32×32	3.6e-3	2.09	1.3e-3	2.00	1.1e-3	2.00
k=2						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	2.3e-0	-	2.6e-1	-	4.3e-1	-
2×2	2.8e-1	3.02	4.1e-2	2.67	4.9e-2	3.12
4×4	3.2e-2	3.13	5.6e-3	2.89	6.3e-3	2.98
8×8	3.8e-3	3.11	7.1e-4	2.97	7.9e-4	2.99
16×16	4.5e-4	3.07	9.0e-6	2.99	9.9e-5	3.00
32×32	5.5e-5	3.03	1.1e-5	3.00	1.2e-5	3.00
k=3						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	3.7e-1	-	5.0e-2	-	3.7e-2	-
2×2	3.3e-2	3.51	3.8e-3	3.73	3.3e-3	3.50
4×4	2.5e-3	3.69	2.5e-4	3.95	2.1e-4	3.94
8×8	1.8e-4	3.80	1.6e-5	3.99	1.3e-5	4.00
16×16	1.1e-5	4.06	9.7e-7	4.00	8.4e-7	4.00
32×32	5.5e-7	4.30	6.1e-8	4.00	5.2e-8	4.00

Table B.8: Stokes VSP: "Hybrid" Mesh, L^2 Error and h-Convergence Rates. Rates are close to optimal.

k=1						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	3.0e-0	-	1.6e-0	-	2.0e-0	-
2×2	1.2e-0	1.30	4.6e-1	1.80	4.1e-1	2.25
4×4	6.2e-1	0.97	1.1e-1	2.12	1.1e-1	1.93
8×8	2.2e-1	1.48	2.7e-2	1.99	2.7e-2	2.03
16×16	6.9e-2	1.70	6.6e-3	2.00	6.6e-3	2.02
32×32	2.0e-2	1.77	1.7e-3	2.00	1.6e-3	2.01
k=2						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	2.1e-0	-	3.3e-1	-	4.6e-1	-
2×2	8.8e-2	4.60	5.2e-2	2.64	5.3e-2	3.12
4×4	8.6e-3	3.35	7.0e-3	2.91	7.3e-3	2.85
8×8	7.7e-4	3.48	8.9e-4	2.98	9.4e-4	2.96
16×16	6.7e-5	3.53	1.1e-4	2.99	1.2e-4	2.99
32×32	6.7e-6	3.32	1.4e-5	3.00	1.5e-5	3.00
k=3						
Mesh Size	p	rate	u_1	rate	u_2	rate
1×1	1.7e-1	-	5.9e-2	-	5.4e-2	-
2×2	1.3e-2	3.74	4.2e-3	3.81	4.0e-3	3.75
4×4	8.9e-4	3.83	2.8e-4	3.93	2.6e-4	3.97
8×8	4.8e-5	4.22	1.7e-5	3.99	1.6e-5	4.00
16×16	2.0e-6	4.59	1.1e-6	4.00	1.0e-6	4.00
32×32	6.7e-8	4.89	6.8e-8	4.00	6.3e-8	4.00

Table B.9: Stokes VVP: "Hybrid" Mesh, L^2 Error and h-Convergence Rates. Rates are close to optimal.

B.3 Solution Plots

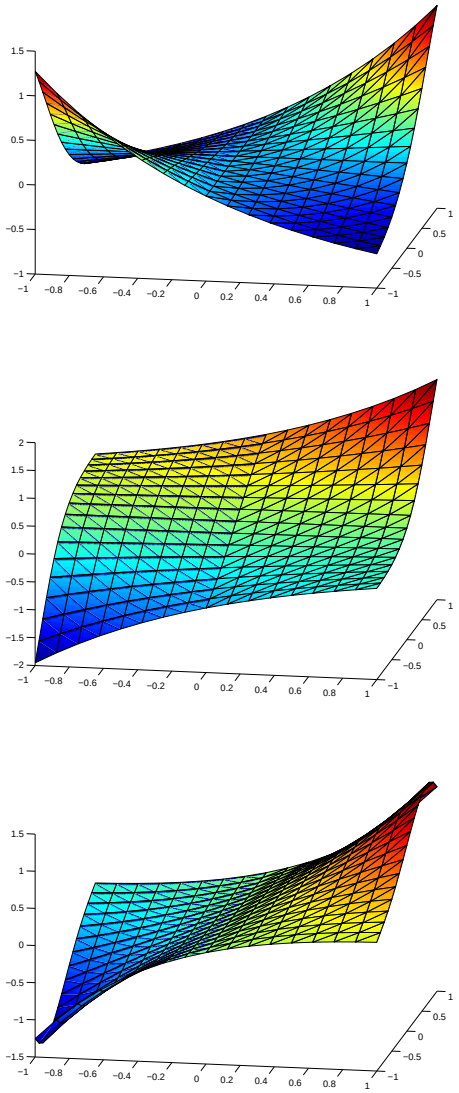


Figure B.1: Solution of Poisson with quads and triangles (cubic elements, 16×16 mesh): ϕ (top pane), ψ_1 (middle pane) ψ_2 (bottom pane).

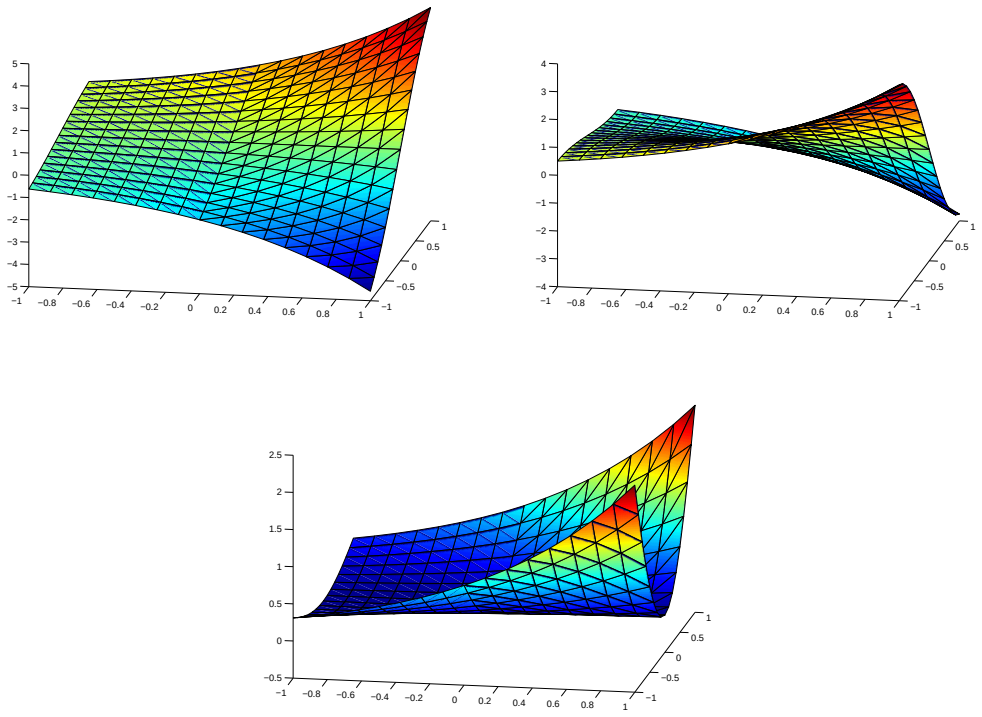


Figure B.2: Solution of Stokes (VVP) with quads and triangles (cubic elements, 16×16 mesh): p (top pane), u_1 (middle pane), u_2 (bottom pane).

Bibliography

[1] N. Alleborn, K. Nandakumar, H. Raszillier, and F. Durst. Further contributions on the two-dimensional flow in a sudden expansion. *J. Fluid Mech.*, (330):169–188, 1997.

[2] B. F. Armaly, F. Durst, J. C. F. Pereira, and B. Schönung. Experimental and theoretical investigation of backward-facing step flow. *Journal of Fluid Mechanics*, 127:473–496, 1 1983.

[3] Wolfgang Bangerth, Carsten Burstedde, Timo Heister, and Martin Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transactions on Mathematical Software*, 38(2), 2011.

[4] Wolfgang Bangerth and Guido Kanschat. Concepts for object-oriented finite element software – the deal.II library. *IWR*, 1999.

[5] Andrew T. Barker, Susanne C. Brenner, Eun-Hee Park, and Li-Yeng Sung. A one-level additive Schwarz preconditioner for a discontinuous Petrov-Galerkin method. `http://arxiv.org/abs/1212.2645`, December 2012.

[6] G. Biswas, M. Breuer, and F. Durst. Backward-facing step flows

for various expansion ratios at low and moderate Reynolds numbers. *Transactions of the ASME*, 126:362–374, 2004.

[7] Pavel Bochev and R. B. Lehoucq. On the finite element solution of the pure Neumann problem. *SIAM Review*, 47(1):55–66, March 2005.

[8] D. Boffi, F. Brezzi, and M. Fortin. Finite elements for the Stokes problem. In *Lecture Notes in Mathematics*, volume 1939, pages 45–100. Springer, 2008.

[9] O Botella and R Peyret. Benchmark spectral results on the lid-driven cavity flow. *Computers & Fluids*, 27(4):421–433, 1998.

[10] C.L. Bottasso, S. Micheletti, and R. Sacco. The discontinuous Petrov-Galerkin method for elliptic problems. *Comput. Methods Appl. Mech. Engrg.*, 191:3391–3409, 2002.

[11] C.L. Bottasso, S. Micheletti, and R. Sacco. A multiscale formulation of the discontinuous Petrov-Galerkin method for advective-diffusive problems. *Comput. Methods Appl. Mech. Engrg.*, 194:2819–2838, 2005.

[12] J. Bramwell, L. Demkowicz, J. Gopalakrishnan, and W. Qiu. A locking-free *hp* DPG method for linear elasticity with symmetric stresses. *Num. Math.*, 2012. accepted.

[13] J. Bramwell, L. Demkowicz, and W. Qiu. Solution of dual-mixed elasticity equations using Arnold-Falk-Winther element and discontinuous

Petrov-Galerkin method, a comparison. Technical Report 2010-23, ICES, 2010.

[14] Susanne C. Brenner and L. Ridgeway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 3rd edition, 2008.

[15] F. Brezzi. On the existence, uniqueness, and approximation of saddle point problems arising from Lagrangian multipliers. *R.A.I.R.O., Anal. Numér.*, 2:129–151, 1974.

[16] F. Brezzi and M. Fortin. *Mixed and hybrid finite element methods*. Springer series in computational mathematics. Springer-Verlag, 1991.

[17] Dirk Broersen and Rob Stevenson. A Petrov-Galerkin discretization with optimal test space of a mild-weak formulation of convection-diffusion equations in mixed form. `http://staff.science.uva.nl/~rstevens/papers/DPG.pdf`, November 2012.

[18] Charles-Henri Bruneau and Mazen Saad. The 2d lid-driven cavity problem revisited. *Computers & Fluids*, 35(3):326 – 348, 2006.

[19] Tan Bui-Thanh, Leszek Demkowicz, and Omar Ghattas. A unified discontinuous Petrov-Galerkin method and its analysis for Friedrichs' systems. *Submitted to SIAM J. Numer. Anal.*, 2011. Also ICES report ICES-11-34, November 2011.

[20] Tan Bui-Thanh and Omar Ghattas. A PDE-constrained optimization approach to the discontinuous Petrov-Galerkin method with a trust region inexact Newton-CG solver. Technical Report 13-16, ICES, 2013.

[21] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[22] P. Castillo, B. Cockburn, I. Perugia, and D. Schötzau. An a priori error analysis of the local discontinuous Galerkin method for elliptic problems. *SIAM J. Numer. Anal.*, 38:1676–1706., 2000.

[23] J. Chan, L. Demkowicz, R. Moser, and N. Roberts. A class of discontinuous Petrov–Galerkin methods. Part V: solution of 1D Burgers and Navier–Stokes equations. Technical Report 25, ICES, 2010. submitted to J. Comp. Phys.

[24] J. Chan, L. Demkowicz, and M. Shashkov. Space-time DPG for shock problems. Technical Report Technical Report LA-UR 11-05511, LANL, September 2011.

[25] J. Chan, J. Gopalakrishnan, and L. Demkowicz. Global properties of DPG test spaces for convection-diffusion problems. Technical Report 13-05, ICES, 2013.

[26] Alexandre Joel Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp*, 22:745–762, 1968.

[27] B. Cockburn, G. Kanschat, D. Schotzau, and Ch. Schwab. Local Discontinuous Galerkin methods for the Stokes system. *SIAM J. on Num. Anal.*, 40:319–343, 2003.

[28] Bernardo Cockburn, Guido Kanschat, and Dominik Schötzau. The local discontinuous Galerkin method for the Oseen equations. *Math. Comp*, 73(246):569–593, 2002.

[29] Bernardo Cockburn, Guido Kanschat, and Dominik Schötzau. A locally conservative LDG method for the incompressible Navier-Stokes equations. *Math. Comp*, pages 1067–1095, 2004.

[30] Bernardo Cockburn, Guido Kanschat, and Dominik Schötzau. The local discontinuous Galerkin method for linearized incompressible fluid flow: a review. *Computers & Fluids*, 34:491–506, 2005.

[31] Bernardo Cockburn, Guido Kanschat, and Dominik Schötzau. A note on discontinuous Galerkin divergence-free solutions of the Navier-Stokes equations. *J. Sci. Comput.*, 31(1-2):61–73, May 2007.

[32] Martin Costabel and Monique Dauge. On the inequalities of Babuška-Aziz, Friedrichs and Horgan-Payne. `http://arxiv.org/abs/1303.6141`.

[33] W. Dahmen, A. Cohen, and G. Welper. Adaptivity and variational stabilization for convection-diffusion equations. *ESAIM: Mathematical Modelling and Numerical Analysis*, 46(5):1247–1273, 2012.

[34] L. Demkowicz. *Computing with hp Finite Elements. I. One- and Two-Dimensional Elliptic and Maxwell Problems.* Chapman & Hall/CRC Press, Taylor and Francis, October 2006.

[35] L. Demkowicz and J. Gopalakrishnan. A class of discontinuous Petrov-Galerkin methods. Part I: The transport equation. *Computer Methods in Applied Mechanics and Engineering*, 199(23-24):1558 – 1572, 2010.

[36] L. Demkowicz and J. Gopalakrishnan. Analysis of the DPG method for the Poisson problem. *SIAM J. Num. Anal.*, 49(5):1788–1809, 2011.

[37] L. Demkowicz and J. Gopalakrishnan. A class of discontinuous Petrov–Galerkin methods. Part II: Optimal test functions. *Numerical Methods for Partial Differential Equations*, 27(1):70–105, 2011.

[38] L. Demkowicz, J. Gopalakrishnan, I. Muga, and J. Zitelli. Wavenumber explicit analysis for a DPG method for the multidimensional Helmholtz equation. *Comput. Methods Appl. Mech. Engrg.*, 213-216:126–138, 2012.

[39] L. Demkowicz and N. Heuer. Robust DPG method for convection-dominated diffusion problems. Technical Report 33, ICES, 2011.

[40] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, and A. Zdunek. *Computing with hp Finite Elements. II. Frontiers: Three-Dimensional Elliptic and Maxwell Problems with Applications.* Chapman & Hall/CRC, October 2007.

[41] Leszek F. Demkowicz. Babuška <=> Brezzi ?? Technical Report 06-08, ICES, 2006.

[42] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[43] Alexandre Ern, Jean-Luc Guermond, and Gilbert Caplain. An intrinsic criterion for the bijectivity of Hilbert operators related to Friedrichs' systems. *Communications in partial differential equations*, 32(2):317–341, 2007.

[44] J. Evans. *Divergence-free B-spline Discretizations for Viscous Incompressible Flows*. PhD thesis, University of Texas at Austin, 2011.

[45] John A Evans and Thomas JR Hughes. Isogeometric divergence-conforming B-splines for the Darcy-Stokes-Brinkman equations. *Mathematical Models and Methods in Applied Sciences*, 23(04):671–741, 2013.

[46] John A Evans and Thomas JR Hughes. Isogeometric divergence-conforming B-splines for the steady Navier-Stokes equations. *Mathematical Models and Methods in Applied Sciences*, 23(08):1421–1478, 2013.

[47] John A Evans and Thomas JR Hughes. Isogeometric divergence-conforming B-splines for the unsteady Navier-Stokes equations. *Journal of Computational Physics*, 2013.

[48] Kurt O. Friedrichs. Symmetric positive linear differential equations. *Communications on pure and applied mathematics*, XI:333–418, 1958.

[49] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[50] V. Girault and P.-A. Raviart. *Finite Element Approximation of the Navier-Stokes Equations*. Springer-Verlag, 1979.

[51] Jay Gopalakrishnan and Weifeng Qiu. An analysis of the practical DPG method. *Mathematics of Computation*, 2012.

[52] W.J. Gordon and C.A. Hall. Transfinite element methods: blending function interpolation over arbitrary curved element domain. *Numer. Math.*, 21:109–129, 1973.

[53] J.L. Guermond, P. Minev, and Jie Shen. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 195:6011–6045, 2006.

[54] J.L. Guermond and P.D. Minev. A new class of massively parallel direction splitting for the incompressible Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 200(23–24):2083 – 2093, 2011.

[55] J.L. Guermond and Jie Shen. A new class of truly consistent splitting schemes for incompressible flows. *J. Comput. Phys.*, 192:262–276, 2003.

[56] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[57] Jiten C. Kalita and Shuvam Sen. Triggering asymmetry for flow past circular cylinder at low Reynolds numbers. *Computers & Fluids*, 59(0):44 – 60, 2012.

[58] G Karniadakis, M Israeli, and S Orszag. High-order splitting methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 97(2):414–443, 1991.

[59] Benjamin S. Kirk, John W. Peterson, Roy H. Stogner, and Graham F. Carey. libMesh: a C++ library for parallel adaptive mesh refinement/-coarsening simulations. *Eng. with Comput.*, 22(3):237–254, December 2006.

[60] L. I. G. Kovasznay. Laminar flow behind a two-dimensional grid. *Mathematical Proceedings of the Cambridge Philosophical Society*, 44(01):58–62, 1948.

[61] L.S.G. Kovasznay. Hot-wire investigation of the wake behind cylinders at low Reynolds numbers. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 198(1053):174–190, 1949.

[62] O.A. Ladyzhenskaya. *The Mathematical Theory of Viscous Incompressible Flows*. Gordon and Breach, London, 1969.

[63] J-L Lions, Yvon Maday, and Gabriel Turinici. A "parareal" in time discretization of PDE's. *Comptes Rendus de l'Academie des Sciences Series I Mathematics*, 332(7):661–668, 2001.

[64] A. Logg, K.-A. Mardal, and G. N. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, 2012.

[65] W.F. Mitchell. A refinement-tree based partitioning method for dynamic load balancing with adaptively refined grids. *Journal of Parallel and Distributed Computing*, 67:417–429, 2007.

[66] H.K. Moffatt. Viscous and resistive eddies near a sharp corner. *Journal of Fluid Mechanics*, 18(1):1–18, 1964.

[67] D. Moro, N.C. Nguyen, and J. Peraire. A hybridized discontinuous Petrov-Galerkin scheme for scalar conservation laws. *Int.J. Num. Meth. Eng.*, 2011. in print.

[68] A.H. Niemi, J.A. Bramwell, and L.F. Demkowicz. Discontinuous Petrov–Galerkin method with optimal test functions for thin-body problems in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 200(9-12):1291–1300, Feb 2011.

[69] J. Tinsley Oden and Leszek F. Demkowicz. *Applied functional analysis*. CRC Press, 2010.

[70] Steven A Orszag, Moshe Israeli, and Michel O Deville. Boundary conditions for incompressible flows. *Journal of Scientific Computing*, 1(1):75–111, 1986.

[71] Rolf Rannacher. Finite element methods for the incompressible Navier-Stokes equations (lecture notes), 1999.

[72] Nathan V. Roberts, Tan Bui-Thanh, and Leszek F. Demkowicz. The DPG method for the Stokes problem. Technical Report 12-22, ICES, 2012.

[73] Nathan V. Roberts, Denis Ridzal, Pavel B. Bochev, and Leszek F. Demkowicz. A toolbox for a class of discontinuous Petrov-Galerkin methods using Trilinos. Technical Report SAND2011-6678, Sandia National Laboratories, 2011.

[74] N.V. Roberts, D. Ridzal, P.N. Bochev, L. Demkowicz, K.J. Peterson, and C. M. Siefert. Application of a discontinuous Petrov-Galerkin method to the Stokes equations. In *CSRI Summer Proceedings 2010*. Sandia National Laboratories, 2010.

[75] Pavel Šolín and Tomáš Vejchodský. Continuous hp finite elements based on generalized eigenfunctions. Technical Report 2006-08, The University of Texas at El Paso, 2006.

[76] Roger Temam. Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires (I). *Arch. Rational Mech. Anal.*, 32:135–153, 1969.

[77] Roger Temam. Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires (II). *Arch. Rational Mech. Anal.*, 33:377–385, 1969.

[78] J. Zitelli, I. Muga, L. Demkowicz, J. Gopalakrishnan, D. Pardo, and V. Calo. A class of discontinuous Petrov-Galerkin methods. Part IV: Wave propagation problems. *J. Comp. Phys.*, 230:2406–2432, 2011.